THE UNIVERSITY OF ALBERTA

# SOFTWARE CONSIDERATIONS FOR A COMPUTER GRAPHICS TERMINAL

by

© Raghupati Sahai Hitkari

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE

OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTING SCIENCE

EDMONTON, ALBERTA

FALL, 1973

ABSTRACT

An investigation of existing systems has been made to determine a choice of hardware for Interactive Computer Graphics, and the software support required for this configuration. This hardware choice was based on two considerations: economic grounds and convenience for the applications programmer in making use of interactive graphics techniques. Much attention has been focused on the software support for the display terminal, an objective being that the applications programmer should rarely be required to write programs for the terminal. This resulted in the development of a multisupervisor scheme for the display terminal. A number of display supervisors have been designed and are described herein.

## ACKNOWLEDGEMENTS

I would like to thank Dr. J.P. Penny for his help and guidance throughout the course of this work. Special thanks to Dr. B.J. Mailloux for his suggestions, and constructive criticism during the writing of this thesis. I am grateful to my wife, Christina, for the typing she did for me.

I wish to thank the Department of Computing Science for the financial assistance and facilities which have made this project possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

# CHAPTER I

## INTRODUCTION

Computer graphics "is playing a significant and an ever-increasing role in the intellectual use of computers as a mechanism for effective communication between humans and machine processes" (Wigington, 1967). Its greatest merit is that it gives the user a diagrammatic representation of his problem. However, it also provides an environment which allows not only for written communication with the computer, but also for human intervention to provide information which is not easily described by means of written statements. The major deterrents to complete acceptance of interactive graphics as a standard tool in computing installations have been the cost and the extensive programming required of the applications programmer for each application. If these difficulties were to be overcome, on-line graphics would become an integral part of almost all computer applications.

Until a few years ago, on-line graphics was done most commonly by means of a direct link between a display unit and a large computer. The dedication of a large computer to control of a single display prevented computer graphics from being economically justifiable. Now, a new trend in computer graphics uses a system in which a cathode ray tube (CRT) display terminal

is linked to a medium-sized computer which is attached to a large, time-sharing computer. The medium-sized computer is used for local servicing of the user's immediate demands on the display, while also communicating with the time-shared computer for complex computations. This thesis, for which most of the work was done in 1968 and 1969, reviews both of these means in their various forms, and discusses their advantages and disadvantages in terms of cost and convenience to the applications programmer. Further, it outlines a hardware choice--a compromise between these two hardware configurations--in which the medium-sized computer is replaced by a much smaller computer, and the graphics computer/CRT is considered to be a programmable terminal. This approach was adopted by the University of Alberta.

Much attention has been focused on the proper exploitation of this configuration in terms of software support for the display terminal. Here a scheme employing multiple supervisors has been incorporated, an objective being that the applications programmer should rarely be required to develop programs for the terminal. Since the terminal is to be used for a wide range of applications, a number of display supervisors are needed, each providing the functional requirements, or the standard "functional behavior" of the terminal, for a particular type of application.

The display supervisors which are thought to be most useful are discussed. These include the "general-purpose supervisor",

the "drawing and editing supervisor", the "text-editing super visor", and the "language processor".

The author believes that the hardware choice discussed herein is sound on economic grounds. He believes also that the software support for it facilitates, for the applications programmer, the use of interactive graphics techniques.

# CHAPTER II

# HARDWARE/SOFTWARE CONSIDERATIONS FOR A DISPLAY CONSOLE SYSTEM

## 2.1 Introduction

Interactive computer graphics has advanced to a stage where a choice of hardware and software approaches relative to many potential economic and practical applications can be made with considerable justification. In this chapter, we review commonly used systems in their various forms, and outline a hardware choice which appears the best on economic grounds. For this configuration, software support is discussed.

## 2.2 Hardware Configurations

The commonest approaches taken in hardware considerations for interactive graphics can be broadly classified under two headings:

(1) Directly-Coupled Displays- a display console (or consoles) directly connected to a medium-to-large computer; or,

(2) Indirectly-Coupled Displays- a display (or displays) coupled to a peripheral computer, which is itself coupled to a larger system.

### 2.2.1 Directly-Coupled Display

For several years, the cathode ray tube (CRT) display console

has been used as a computer input/output device that could accept or exhibit data in both pictorial and alphanumeric forms. The display console was directly connected to, and used the total resources of, a medium-to-large sized computer. The core memory held the applications program, display file, and routines for vector and symbol generation and pen tracking. Figure 2-1 shows such a system. In 1963, the SKETCHPAD system (Sutherland, 1963) was designed and implemented by I.E. Sutherland on the TX-2 computer in this general way.

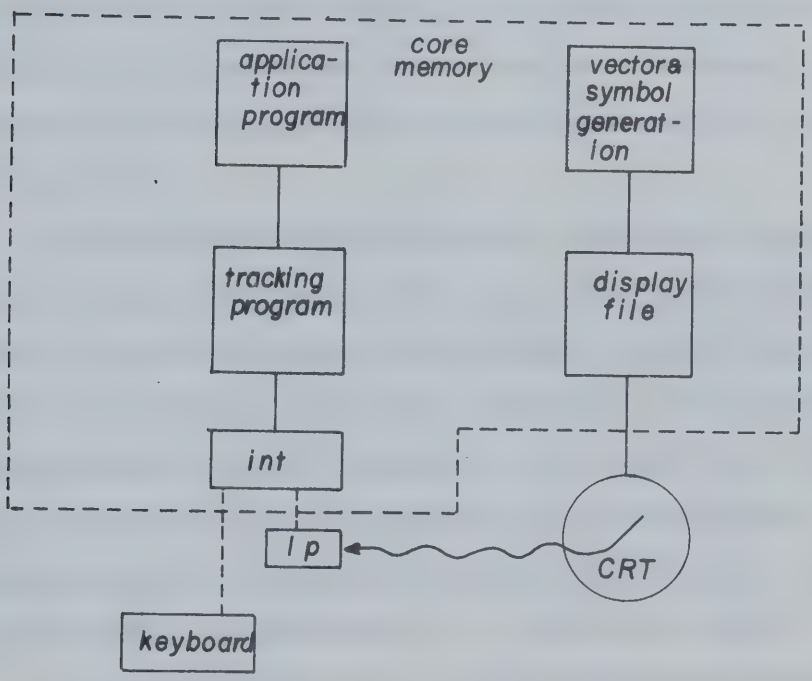



Figure 2-1

In this older system most of the functions were programmed and minimal hardware was used. Direct connection through an I/O channel gave rapid access to the applications program; thus, the system lent itself well to experimentation. However, the cost involved was high, due to the dedicated use of a large computer to control a single display.

The very essence of this system, the fact that a large computer was directly linked to and used by, a single user was not justified. Since every hour of console time required only a few minutes of main computer time, much time during which the computer could be used for other work was not utilized. Fortunately, the wider availability of time-sharing systems has since overcome this problem.

Another shortcoming was that display regeneration required both space and time on the computer, and later display systems usually included independent buffer storage. Display instructions, such as "enter vector mode", "enter point mode", "enter character mode", and the data indicating x-y positions of lines and characters are stored in digital form in this buffer for display generation. Figure 2-2 shows the block diagram of the IBM 2250 MODEL I system (Appel et al, 1968) as an example.

Display regeneration from the storage buffer conserved the core storage of the large computer, thus allowing for more

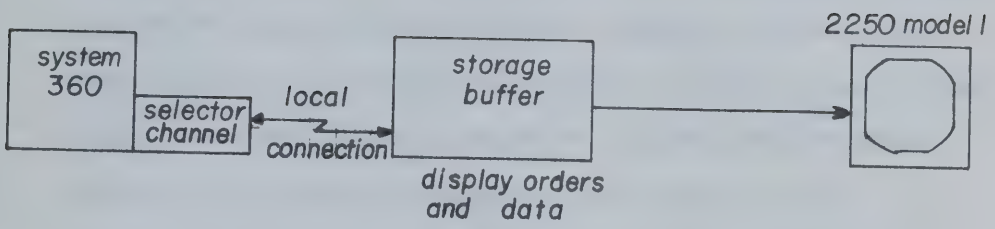



Figure 2-2

extensive application programs. Although a buffered display is less demanding on the CPU than a directly-coupled display, the equipment is still expensive since a considerable amount of hardware is used merely to regenerate images. Furthermore, the majority of requests issued by a console user require only trivial processing, such as the display of a typed character or a reaction to a light pen pick. These could be done by a small computer.

This idea of a smaller computer being used with a display console brings us to the concept of an indirectly-coupled display.

### 2.2.2 Indirectly-Coupled Display

In the indirectly-coupled display system the terminal is linked to a medium-sized, general-purpose computer which is attached to a large, time-shared computer. The graphics computer is used for local servicing of minor requests, while also communicating with the time-shared computer for complex computations. Figure 2-3 shows the block diagram of the IBM 1130-2250 MODEL IV (Appel et al, 1968) as a representative of such a system. In this configuration, the 1130 CPU is used for highly interactive graphics operations plus display regeneration, leaving the host (IBM 360) to deal with only occasional major requests.

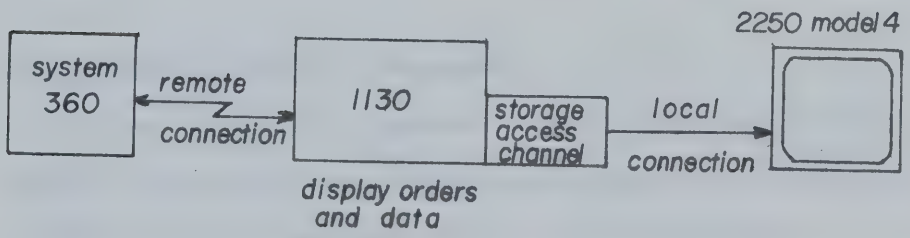



Figure 2-3

Although the indirectly-coupled display system significantly reduces the conversational overhead on the large, time-sharing

host system, it is particularly expensive. The terminal and the graphics computer are monopolized by a single user, and the graphics computer itself is not really small. More importantly in the present context, to use a dual computer system, the applications programmer has to develop programs for both computers. This requires much programming knowledge which is not commonly expected of a typical applications programmer.

Since we are discussing hardware considerations per se, it is useful to consider an approach that compromises between the buffered display system and the indirectly-coupled display system. This approach provides a base convenient to the applications programmer and is more economical than a configuration such as the IBM 1130/2250 coupled to a Model 360.

### 2.2.3 Hardware Choice--A Compromise

We can replace the medium-sized computer by a much smaller# computer and consider the graphics computer/CRT as a programmable terminal. This approach was adopted at the University of Alberta, where a Control Data 160A/GRID graphics terminal has been coupled to an IBM 360 Model 67. With this configuration, the applications programmer is required to write his programs for one computer--the 360--only.

---

# The basic cost of the Control Data 160A/GRID graphics terminal is 90,000 dollars, and its maintenance cost per month is 450 dollars; whereas, the basic cost of the IBM 1130/2250 is 132,600 dollars, and its maintenance cost per month is 900 dollars.

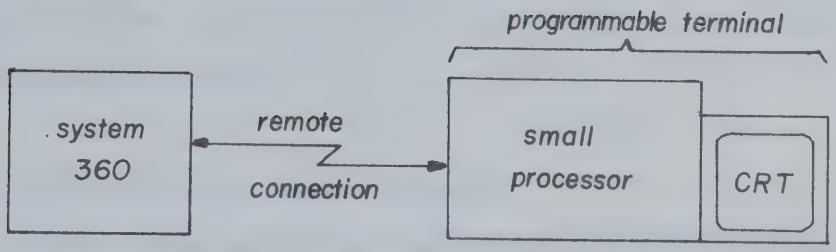



Figure 2-4

Reasonably convenient exploitation of such a programmable terminal system is possible only if we rarely, if ever, require the applications programmer to develop programs for the terminal. For the University of Alberta system, a small number of standard, "functional behaviors" have been defined for the terminal depending upon the context in which the console is to be used, and the applications programmer can specify how he wants the terminal to behave, that is, which of the standard behaviors he wants, and can change this whenever required. The behavior of the terminal is determined by a display supervisor, and the FORTRAN display-generating routines assemble a display file which interfaces with any supervisor, regardless of its length. This idea was adopted by Huen et al (Huen et al, 1969). However, only one supervisor

was actually developed, and the idea was not fully exploited. In later chapters, we describe other standard behaviors which seem worth having available.

## 2.3 Software For The Display Terminal

In providing the software support for the display terminal, the following objectives are important:

(1) There must be a clearly defined software interface between the two computers.

(2) The number of interruptions to the main processor should be minimized.

(3) The system should be interrupt-driven.

(4) The software should be such that the display terminal can be used as a stand-alone system for on-line work which involves the generation of frequent interrupts.

(5) The software should be open-ended and, thereby, allow for future modification and extension.

Taking each of the above objectives in turn, we now discuss their advantages.

One of the most important considerations is the need to provide a clearly defined software interface between the two processors. That is, there should be a task division between the display processor and the main processor. For example, let us suppose that all picture development is done in the main processor. We must

allow the user to develop quite large pictures of which only small parts are seen on the CRT (and held in the display computer) at any given time. The description of the picture might be stored in a compound data structure (Sutherland, 1963), (Gary, 1967). If the user wishes to see a different part of the picture, his request must be transmitted to the main computer and it must select the appropriate parts of the data structure and generate a "display file" of line, point, and character drawing commands. This display file can then be transferred to the display processor.

Requests to show a different portion of the picture may be expected to be rare relative to the total number of interrupts--rare enough to suggest that they can always be referred to the main processor. If we go further and say that all changes (additions, deletions, modifications, etc.,) which are required to alter the picture must first be performed on the data structure in the main computer, followed by regeneration (or editing) of the display file, we then can have an extremely clear interface.

We now turn our attention towards meeting objective (2). In order to reduce the overhead associated with interrupt-servicing, the number of interrupts to the main processor should be reduced to a minimum. Therefore, not every attention interrupt by the console user should be routed to the main processor. Some types of on-line work involve the generation of a large number of

interrupts. For example, during pen tracking, interrupts may occur at a rate of 50 per second. Most of these interrupts can be processed by the small processor, leaving the main processor to deal with only the occasional major request.

We could continue to provide examples, but suffice it to say that many fairly simple operations requested by the user can be dealt with by the display processor. However, when a major request, such as for changes to the picture, is made, then this request must be encoded and sent to the main processor so that the data structure can be amended.

In an interactive-graphics system, the operator at the console communicates with the system through actions (pen picks, pressing function keys, and so on) each of which results in an interrupt. Since each of these interrupts has some definite meaning, the system must be able to deal with as many interrupts as can possibly occur during the time a user is waiting for his next action. The applications program may then process the attentions when the system resources (e.g. CPU) are available. Moreover, the ability of the display console to display much information rapidly makes the console user impatient; he wants to be assured by some feedback that the system has recognized his request. The system should be interrupt-driven--at least to the degree that no attention interrupts

are lost and a rapid acknowledgment is returned on the user's display to indicate that the system has received his request.

The software support should make the display terminal able to work as a stand-alone system for some problems. For example the drawing of lines, which involves the generation of frequent interrupts, should be performed by the display processor as a stand-alone system.

The justification for objective (5) is obvious.

As mentioned earlier, the display supervisor determines the behavior of the display terminal; therefore, a multiple supervisor scheme has been adopted, with each supervisor providing one of the standard behaviors (discussed in the next Chapter) of the terminal. Under this scheme there are a number of display supervisors. Among these, one is a "general-purpose" supervisor to provide a variety of general functions; others are "special-purpose" supervisors, each specific to a class of application.

# CHAPTER III

# STANDARD BEHAVIOR(S) FOR THE DISPLAY TERMINAL

## 3.1 Introduction

In the last chapter, we discussed hardware configurations in terms of their advantages and disadvantages. We established that the best hardware choice at the present time is one in which the graphics computer plus the CRT are considered a programmable terminal.

The proper utilization of this system requires that software for the display terminal be provided for the applications programmer. Ideally, the applications programmer should have to write programs only for the main CPU and in only one language. Therefore, we must provide the requirements, or the "functional behaviors", for a range of applications programs.

Most frequently, the terminal functions as an input device, accepting (and showing responses to) messages. However, some applications require additional functional behaviors. For example, let us consider as an applications program the Computer Aided Logical Design (CALD) System (Johnson, 1969), which allows a designer to construct and modify a circuit design at the console. As far as he is concerned, the display screen is divided into two areas: The control area on the left and the display area on the right, as shown in figure 3-1.

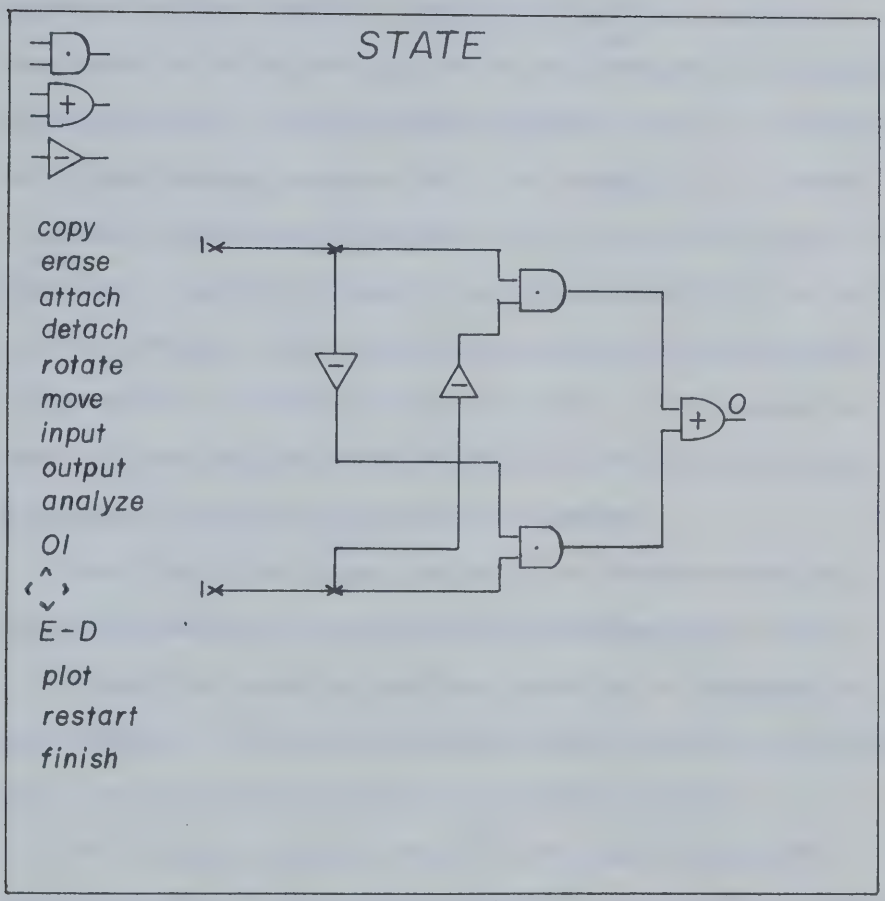



Figure 3-1 A CALD display. (Courtesy of B.V. Johnson)

The circuit designer can indicate the operation desired by pointing the light pen at a command word and the logical components, or by depressing a key on the keyboard. His actions are governed by a "console command language" which is specified by the applications program, CALD. For example, if he wishes to display an AND gate at position (x,y) in the display area, he points the light pen in turn at the command word COPY, the logical component AND gate, and to the (x,y) position at which the gate is to appear. If, through error, he points to the command word COPY and then indicates with the light pen the (x,y) position, he expects the system to give an error message.

Thus, as an additional feature of the message-accepting functional behavior, the applications programmer might expect the terminal to contain an analyzer, checking all messages for syntax errors. It would, therefore, transmit only those messages which are syntactically correct to the main computer.

To illustrate the fact that different applications can require additional functional behaviors, let us consider another example, campus planning (Hough, 1968), (Deecker, 1970). Deecker's system is designed to help the user to choose the optimum site for a new building. Maps of the campus are stored in the computer; and maps, or parts of maps, can be retrieved for viewing in any combination, and at any desired scale.

For the purpose of this application, it would be useful if there were a supervisor with an extended version of the message-accepting behavior to include line drawing and editing capabilities on the screen. This would allow the console user to develop plans with freehand drawing.

Because different applications often require different functional behaviors, we have two significant problems:

(1) Deciding what functional behaviors for the terminal are desirable as standard; and,

(2) Deciding how they can best be provided.

A solution to each of these problems is the subject of this Chapter.

## 3.2 Software Considerations for the Terminal

First let us deal with the latter problem, that of developing software support to provide for standard functional behaviors.

The behavior of the terminal is determined by a display supervisor (Huen et al, 1969). By developing a supervisor for the terminal which provides all the functional behaviors needed, the terminal can be used to perform actions for which the main computer would otherwise be required. The advantages are better response time, and more economical use of the main computer. However, the approach of including all functional behaviors in a single supervisor is not feasible due to the memory limitations

of the small computer. Furthermore, of the variety of functions provided in such a display supervisor, many may not be required by any one applications program. Thus, the software to perform them all would unnecessarily occupy space in the display terminal, leaving less space for other co-resident programs, such as the input and output routines, and the display file. Therefore, as an alternative, we must have a number of display supervisors, each providing one of the standard functional behaviors for the terminal.

### 3.2.1 Display Supervisors

A multiple supervisor scheme has been adopted. Under this scheme there are a number of display supervisors, each supervisor providing one of the standard behaviors of the terminal. Among these supervisors, one is a "general-purpose" supervisor to provide the general behavior needed by the applications programmer, and others are "special-purpose" supervisors, one for each specific class of application.

Usually, the general-purpose supervisor resides in the display terminal. Other supervisors must reside in the auxiliary memory of the main processor because their frequency of use is assumed to be low. With a dynamic load-at-call-time capability, the supervisors which are required by the display user can be shuttled into

and out of the display terminal on a demand basis. However, during the operation of a single applications program, the general-purpose supervisor would be used all, or most, of the time.

The display supervisors thought to be most useful are discussed in the following sections.

## 3.3 General-Purpose Supervisor

At the University of Alberta, the original general-purpose supervisor was developed by Penny (Jacobsen et al, 1970) for a GRID system with only one bank of core memory. Later on, it was extended to a multibank GRID system by Sulkers, Smith, and Zavitz (Sulkers et al, 1971).

The primary function of the general-purpose supervisor is to assemble a sequence of actions by the console user into a message for transmission to the 360. It contains routines for handling interrupts and maintaining the display. At its nucleus is a program which analyses and interprets the inputs. This program, the "manual interrupt processor" is table-driven. The three tables to which it refers are known as "interrupt tables". Each table is for a specific type of interrupt--one for light pen interrupts, another for function key interrupts, the third for alphanumeric keyboard interrupts.

The interrupt tables for the function keys and the alphanumeric keyboard contain entry points to the "interrupt processing routines". The interrupt table for the light pen is arranged as pairs of entry points and buffer addresses. Whenever a light pen interrupt occurs, the interrupt address is compared to the buffer address to find the entry point.

When a manual interrupt occurs, the manual interrupt processor first determines which of the three types it is. Then, after referring to its associated interrupt tables, it branches into one of a number of interrupt routines.

The use of interrupt tables permits easy modification of the supervisor since the tables can be readily extended, rearranged, or changed. Therefore, the general-purpose supervisor is a standard supervisor framework from which different supervisors can be built simply by adding or replacing interrupt routines, and by changing the entries in the interrupt tables.

## 3.4 Drawing and Editing Supervisor

The drawing and editing supervisor has been designed and implemented by the author to do freehand sketching and subsequent refining of arbitrarily shaped curves on the display. With the light pen, the designer is able to create and modify line drawings at will. The drawing and editing supervisor, which was designed

to be permanently resident in the auxiliary memory of the main processor, can be loaded into the GRID by a FORTRAN subroutine call. The programmer can make this call by either assigning a function key or pointing to the command word "draw" with the light pen.

The supervisor provides for three modes of operation. One, the draw mode, allows the console user to draw arbitrary lines on the display with the light pen. Another, the edit mode, is used solely to edit existing line drawings by means of function keys in conjunction with the light pen. The third, the track mode, is used when indicating positions, such as the starting point of a line segment and the endpoints of the portion of a line to be erased. The drawing supervisor is described in detail in Chapter IV.

## 3.5 Text-Editing Supervisor

A text-editing supervisor has been designed (though not implemented) by the author to allow the console user to input text and to edit text at the console. The editing of text would be done through the use of the light pen and a set of function keys. The editing functions would be selected by pressing an appropriately labelled key. The portion(s) of text to which the function is to be applied would then be indicated by pointing at the text with the light pen. For example, to delete a portion of

the text, the "delete" key would be pressed, after which the characters delimiting the text to be deleted would be pointed at with the light pen. The indicated text would then disappear from the display.

The advantages of editing with function keys and the light pen are that no command codes for the edit functions need be remembered, and no typing is required to indicate a context string.

The text-editing supervisor, described in detail in Chapter V, would have three modes of operation--input, edit, and command modes. Input mode would be used during input of text; edit mode for editing of existing text; and other modes for communication with the main computer.

## 3.6 Language Processor

To communicate with the 360, the display operator at the console has several means of constructing a message. For example, he can pick displayed items with the light pen, press function keys, type strings of alphanumeric characters, indicate positions on the screen, or use any combination of these actions. For each application, a "console command language" can be defined which specifies the sequences of actions to be followed when constructing a message for that particular application. Since the console user might make an error in the construction of a message,

an extended version of the general-purpose supervisor can be designed to check for syntax errors. The specification of a message-checking supervisor is given in Chapter VI.

# CHAPTER IV

# DRAWING AND EDITING SUPERVISOR

## 4.1 Introduction

One of the most useful display supervisors is the drawing and editing supervisor, which allows the console user to draw and edit at the display scope at will. The drawing and editing supervisor could also, with minor changes, be used with some other line drawing devices such as the RAND tablet (Davis and Ellis, 1964).

Drawing or tracking with the light pen causes interrupts to be generated once per display cycle--that is, at the rate of 50 per second. With such a frequency, it is essential that these interrupts be handled by the display terminal without reference to the main processor. This, in itself, implies a need for the drawing supervisor.

In this Chapter, we describe the structure of this supervisor, and the behavior of the display terminal under its supervision.

## 4.2 General Structure of the Supervisor

The drawing and editing supervisor (hereafter called "the drawing supervisor") has been designed and implemented to allow the console user to draw and to edit arbitrary line drawings. The

supervisor utilizes the capabilities of both the display terminal and the 360. The duties of the 360 are fairly limited. They include the maintaining and storing of files, sending of the display files, and reorganization of the display files. Handling of input and commands, and transient editing is all done in the GRID. The advantages of this partition are immediate response to the user's actions and more economical use of the 360.

The duties of the GRID include accepting input at the console, displaying the contents of the drawing buffer, displaying a display file transmitted from the 360, and editing line drawings. The organization of storage in the GRID during operation of the drawing supervisor is shown in figure 4-1.

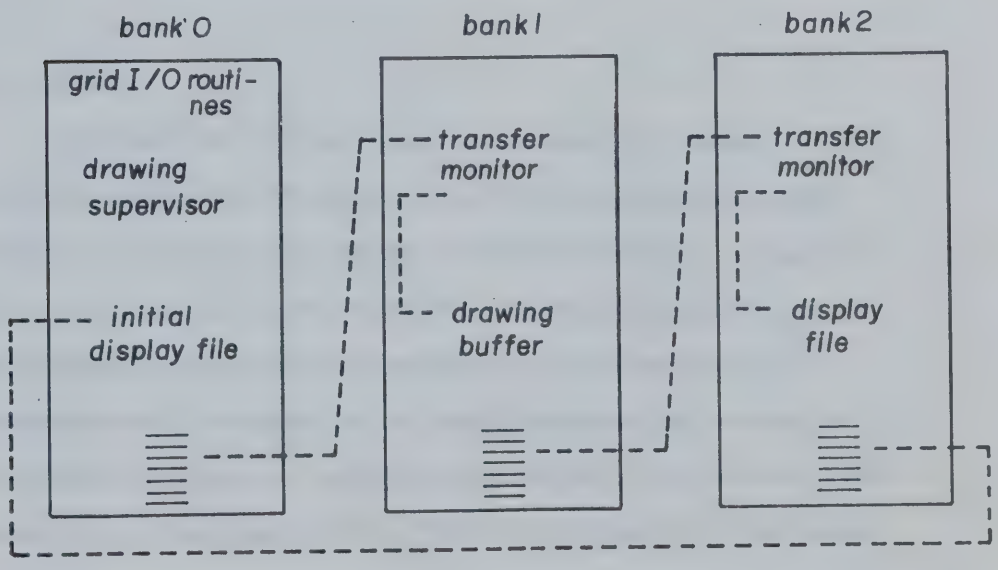



FIGURE 4-1

The drawing supervisor has four modes of operation, invoked by using a specific function key for each mode. The four modes are:

(1) The "draw" mode, which is used solely for arbitrary line drawing on the display.

(2) The "track" mode, which allows the console user to do light pen tracking, without line generation.

(3) The "edit" mode, which allows erasure and insertion of lines in drawings already on the screen.

(4) The "command" mode, which is used to send specific orders to the 360 regarding editing changes to be made in the stored files.

## 4.3 Drawing

The "draw" mode of the supervisor allows the drawing of arbitrary lines with the light pen. To indicate the starting point of a line, the user brings a "tracking matrix", by means of light pen tracking (to be discussed), to the desired position and presses the function key assigned as the "draw" key. The coordinates of the origin of the tracking matrix are stored in the display computer to indicate the starting point of that particular line. The command word, "DRAW", now appears on the screen to indicate that the user is in the drawing mode, and any movement

of the tracking matrix with the light pen leaves a solid line (actually a sequence of straight line segments) following the course of the light pen.

Each depression of the "draw" key corresponds to a separate line. A series of lines can be drawn until the buffer is full, at which time the contents of the drawing buffer are automatically transferred to the 360. Alternatively, the user can send the contents of the drawing buffer to the 360 at any time by pressing the "send" key. Upon receipt, the contents of the drawing buffer may be analyzed by using an assembler language routine, DDVECT (Appendix A) designed and implemented by the author. This routine, callable from a FORTRAN program, provides the console user with information, such as the number of lines drawn, the number of points in each line, and the starting point of each line, by setting a number of arrays. Using the information stored in these arrays, the user can use GRIDSUB routines to add this line drawing to his permanent drawing files and/or to the display file.

Line segments are encoded by storing their end points. The position of the tracking matrix is updated once per display cycle, that is, about 50 times per second. However, the console user can designate how frequently the updated origin of the tracking matrix should be stored, simply by typing the appropriate number on the keyboard. For example, if the console user types the number "6",

every sixth updated origin of the tracking matrix is stored (figure 4-2).

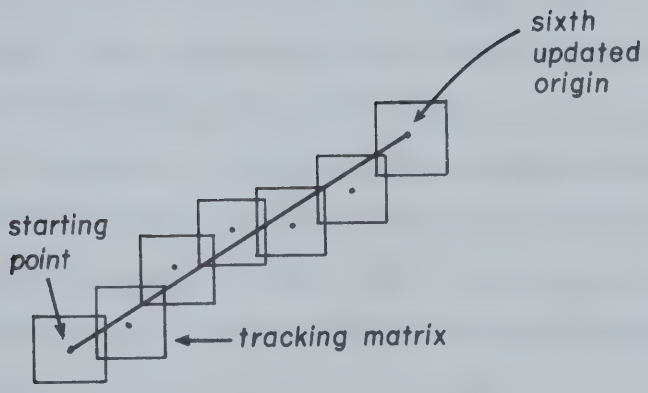



FIGURE 4-2

If the user specifies no particular frequency, every fourth updated origin is stored. If he specifies every tenth updated position, this would have the effect of smoothing the line, at the cost of accuracy. However, he can alternatively achieve a greater amount of accuracy, by requiring that more frequent origins be stored, at the expense of storage space.

## 4.4 Light Pen Tracking

A variety of methods exist for input of positional information

on a display screen. Special-purpose input devices, such as a drawing tablet (Davis and Ellis, 1964) are frequently used. Since a graphical display almost invariably has a light pen for entity picking, use of the light pen for coordinate input is obviously desirable. Since a light pen can only sense light, use of it to indicate positions on blank screen areas or to draw lines requires special techniques. A "raster scan" technique employed by Huen et al (1970) involves vertical and horizontal scanning of the screen with software-generated vectors until the position at which the pen is pointed is found. Other techniques involve "tracking", or following, the pen with a pattern of points.

For a system requiring both input of points and drawing of lines, tracking methods are necessary. The author experimented with two tracking methods which are described below.

### 4.4.1 Centroid Method

A common tracking technique is the "centroid" method. The centre of the field of view of the light pen is found by getting four interrupts, one from each arm of a set of points displayed in the form of a cross (figure 4-3). Arms of the cross--a,b,c,d--are displayed in turn, each as a sequence of points from its outer end to the origin ($X_0$, $Y_0$). If the light pen detects a point on arm 'a', the X coordinate value ($X_1$) is stored and display of arm 'b' is commenced.

If a point is detected on arm b, the Y coordinate value ($Y_2$) is stored. If four interrupts are detected on one display cycle, a new origin,

$$(X_0\ Y_0) = \frac{X_1 + X_3}{2}, \frac{Y_2 + Y_4}{2},$$

the centroid of the field of view, is calculated, and the cross is moved to this position. In this way, the cross follows the movement of the pen.

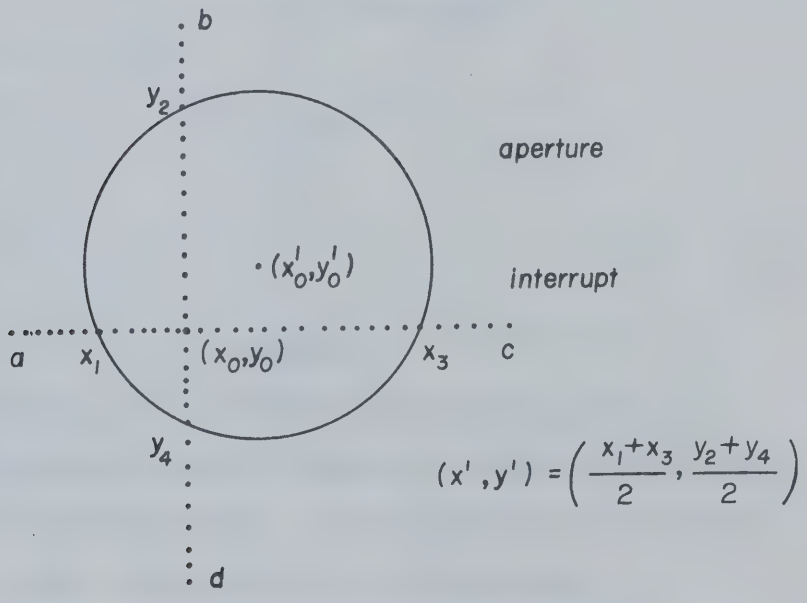



FIGURE 4-3

The author found that use of the centroid method allowed accurate following of the pen; however, the speed of pen movement is restricted for reasons to be explained by reference to figure 4-4.

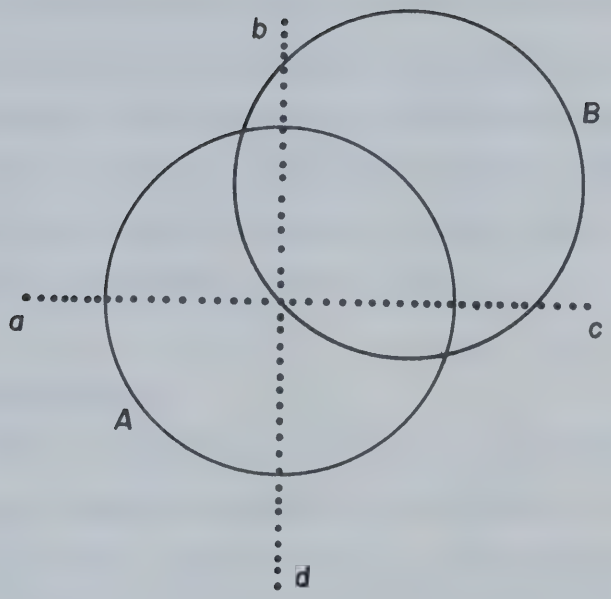



FIGURE 4-4

If the light pen is moved the short distance from position A to position B between successive displays of the cross, the pen cannot "see" points on all four arms. The user must then move it back to pick up the cross, thus impeding the tracking process.

This problem arises if the pen is moved a distance r within t milliseconds, where r is the radius of the field of view and t is the interval between successive display cycles. For the GRID display and light pen, r = 0.05" and t = 20 msec. That is, the pen must be moved at speeds less than about 2.5" per second. The interval between successive displays cannot be substantially decreased.

Use of a light pen with a wider aperture would help, but it would then be unsuitable for picking individual characters in a text string. Light pens with adjustable apertures have sometimes been used for the dual function of picking and tracking. However, the author feels that these are inconvenient to use, and he has therefore developed an alternative tracking method.

4.4.2 Scanning Method

The method of tracking chosen for use in the drawing supervisor is known as the "scanning" method. In this method, the tracking cross is replaced by a matrix of points. The tracking matrix, with dimensions of 1/4 by 1/4 inches, consisting of seven rows of seven points, is generated twice horizontally, with its origin at the point $(X_0, Y_0)$, as shown in figure 4-5.

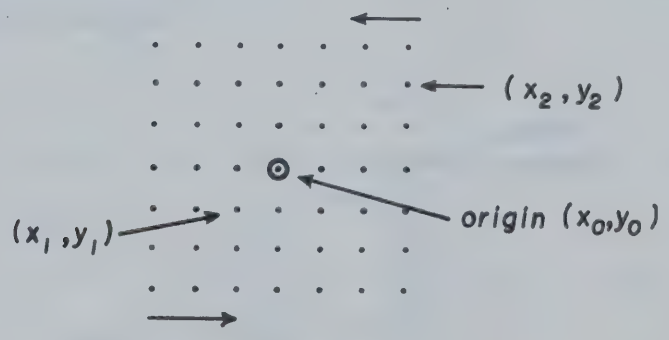



FIGURE 4-5

First, all rows of the matrix are generated in sequence, starting with the lowest row and generating each towards the right, until a light pen interrupt is obtained, and the coordinate values X, and Y, of the point are read. The generation of rows is then repeated, this time starting with the uppermost row and generating each towards the left, to get the second required light pen attention. Assuming that the coordinate values obtained from the second light pen attention are $X_2$ and $Y_2$, the coordinates $X_0'$, $Y_0'$ of the new origin are calculated by

$$X_0' = \frac{X_1 + X_2}{2} \quad , \quad Y_0' = \frac{Y_1 + Y_2}{2}$$

Once the new origin is calculated, the matrix is again displayed at this position in order to get the next pen position.

The reason for generating the matrix in this fashion is that the points detected will be on the circumference of the field of view, as shown in figure 4-6. Therefore, a bigger displacement of the origin is obtained.

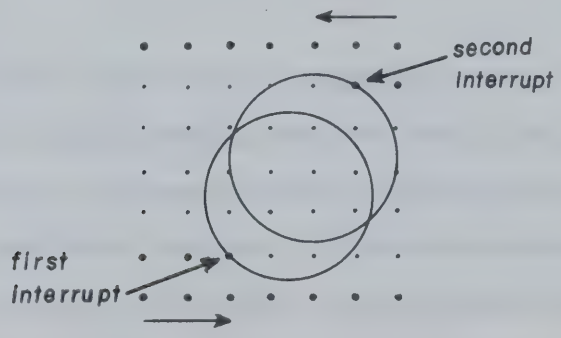



FIGURE 4-6

The scanning method of tracking allows for faster movement of the light pen than the centroid method; because, even with the displacement illustrated in figure 4-4, the pen can still see at least some points of the matrix, as shown in figure 4-7.

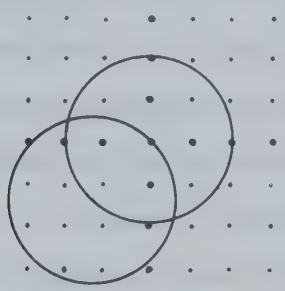


FIGURE 4-7

It is obvious that the pen can move up to about 0.125" between successive display cycles without "losing" the tracking matrix. The pen can therefore be moved at rates up to 6.2" per second. This is in contrast to the centroid method with which the pen must be moved at speeds less than 2.5" per second.

## 4.5 Editing

There are several possible ways of handling editing. Ideally, as much as possible of the editing should be done in the display terminal for the sake of economy and for better response time. In this drawing supervisor, editing of the temporary display file or the contents of the buffer (the editing most frequently required)

is done in GRID, leaving the 360 for occasional editing of the permanent display files and for the garbage collection which is necessary as a result of editing.

To illustrate the edit mode, consider an example in which the console user wants to erase a portion of a drawing and insert a different line in its place. To do this, he puts the display in the edit mode by pressing the function key assigned as the "edit" key; then, with the light pen, he indicates the end points A, B of the portion of a line to be erased. The supervisor displays a tracking matrix at each position A and B (figure 4-8). Next, the user presses the ERASE key. The portion of the line between A and B disappears, and the tracking matrix at point B is deleted (figure 4-9). To insert a line, the user presses the INSERT key to put the display in "insert" mode. He then uses the light pen to draw the desired line. Following this, he presses the ENDEDIT key which connects the center of the tracking matrix at the end of the inserted line to point B, as shown in figures 4-10 and 4-11.

The contents of the buffer before and after editing are shown in figure 4-8 and 4-11 respectively. No garbage collection is performed in GRID. When the drawing buffer is full, its contents are transferred to the 360, which carries out garbage collection. Since use of the garbage collector is relatively infrequent, providing it in the 360 rather than GRID presents no significant inconvenience to the user.

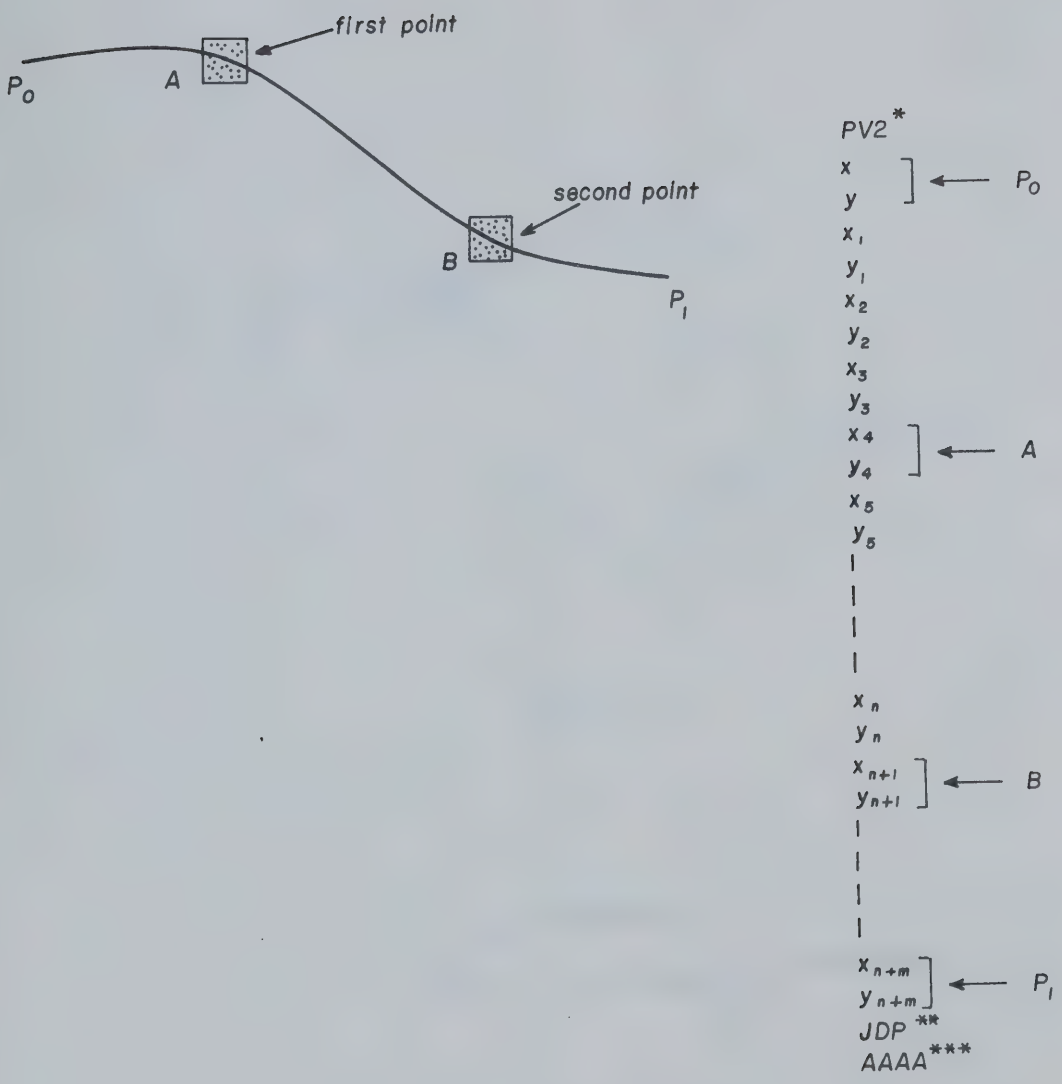



FIGURE 4-8

---

* Mnemonic for Plot Vector Mode 2

** Mnemonic for Jump Display Processor Mode

*** Memory Address

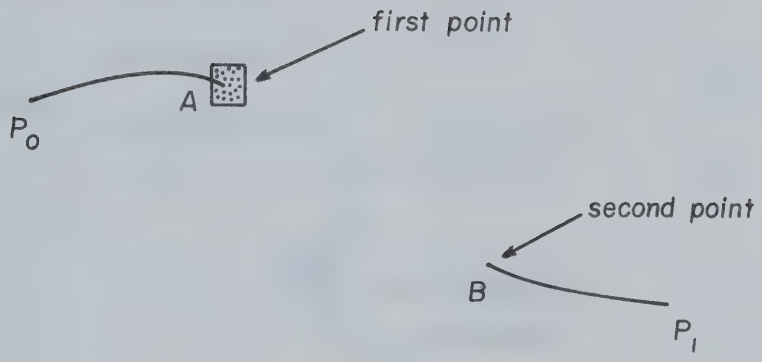



FIGURE 4-9

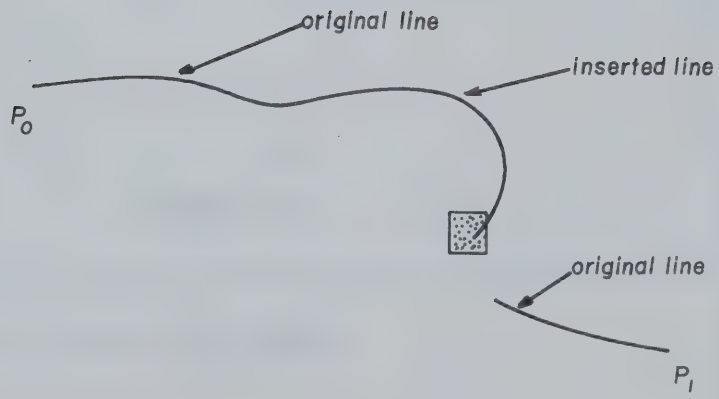
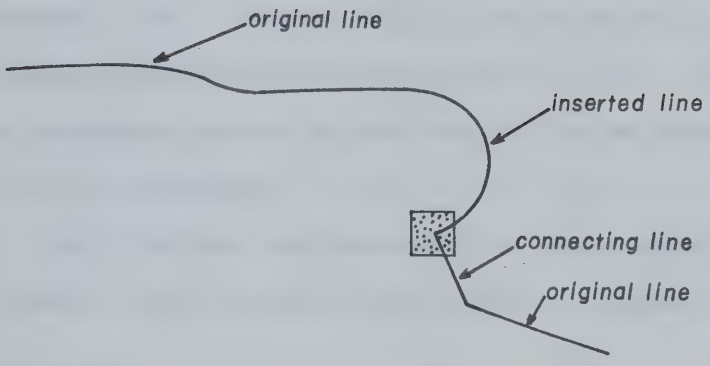



FIGURE 4-10



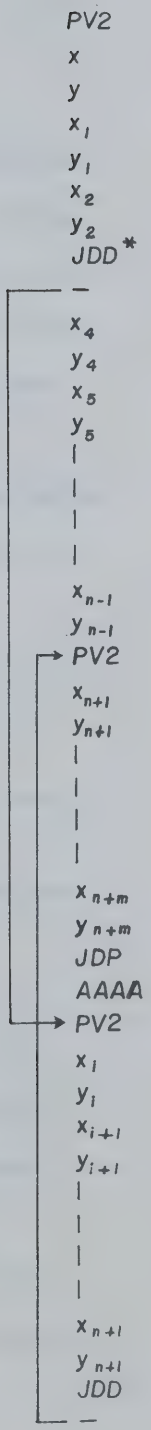




FIGURE 4-11

---

* Mnemonic for Jump Display Display

4.5.1 Editing Intersecting Lines

Editing of intersecting lines should be performed in a different manner than the method that has been described in the previous section. This is because, in the case of intersecting lines, the user must specify which line he wishes to erase. He would do this by designating a point on the intended line and erasing the line as two line segments.

To illustrate this procedure, let us suppose that a user wants to erase "Line 1" between points A and B, as shown in figure 4-12.

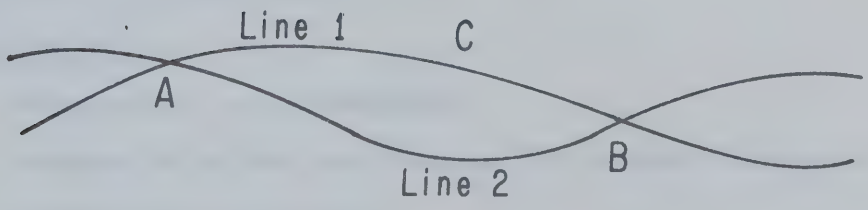



FIGURE 4-12

He would do this by considering another point, say C, on Line 1 and then erasing the line segments AC and CB following the procedure described in Section 4.5.

Now, if the user wants to insert another line in place of Line 1, he would follow the insertion procedure described earlier.

## 4.6 Display Terminal Behavior

The behavior of the display terminal under the drawing supervisor can be divided into two distinct phases:

Phase 0-- drawing and editing phase

Phase 1-- message-accepting phase

Phase 0 allows the console user to draw arbitrary lines on the display, and to edit existing drawings on the screen. The operations performed in this phase by the light pen, function keys, interrupt key, and send key are explained below.

Light Pen:

The light pen is used for three operations: drawing, tracking, and picking a specific item.

Function Keys

Function Key 1-- the BUFSEND key

Pressing the BUFSEND key causes the I/O routines to send the drawing buffer to the 360.

Function Key 3-- the DRAW key

Pressing the DRAW key causes the display terminal to enter the "draw" mode. Now, by using the tracking matrix and the light pen, lines may be drawn on the display.

Function Key 4-- the RESET key

Upon pressing the RESET key, the line so far generated is deleted without any transmission to the 360.

Function Key 5-- the EDIT key

When the EDIT key is pressed, the display terminal enters the "edit" mode. The line drawings are enabled, that is, made light-pen detectable.

Function Key 6-- the ERASE key

The portion of a line drawing which lies between two tracking matrices disappears when the ERASE key is pressed.

Function Key 7-- the TRACK key

The TRACK key allows the tracking matrix to be moved without line generation.

Function Key 8-- the INSERT key

The depression of the INSERT key allows an insertion to be made where an erasure has already been made.

Function Key 9-- the ENDEDIT key

The ENDEDIT key, once pressed, connects the inserted line to the end of the remaining line.

Function keys 0 and 2 have no effect.

Interrupt Key-- JOB END

Upon pressing the INTERRUPT key, the words JOB END will appear at the top right-hand corner of the screen. However, the job will not terminate until the message has been transmitted to the 360.

Send Key-- SEND

Pressing the SEND key causes the I/O routines to send the message which consists of the actions taken by the console user in order

to modify the display file to the 360.

Phase 1 allows the console user to collect the information in the form of a message to be sent to the 360 for editing of the display files. In Phase 1, any operator action which necessitates changes in the display file is recorded in the message to be sent to the 360. However, nothing is sent to the 360 until the SEND key is pressed. The message-accepting phase of the drawing supervisor is the same as in the "general-purpose supervisor" (Huen et al, 1970).

# CHAPTER V

# TEXT-EDITING SUPERVISOR

## 5.1 Introduction

In the previous chapter, we discussed the drawing and editing supervisor which allows the user to draw arbitrary line drawings and to edit existing drawings at the display console. Another display supervisor which would be desirable under the multisupervisor scheme for the GRID is a text-editing supervisor. The present chapter justifies the need for such a supervisor for GRID, describes the general structure of a proposed supervisor, and possible means of implementing it.

## 5.2 Text-Editing and On-Line Graphics

The current methods of computer-assisted text-editing (Bourne, 1971), (Deutsch and Butler, 1967), (Carmody et al, 1969), (Elliot et al, 1971) have in some areas replaced the traditional secretarial editing method. This is because the traditional method of editing requires many cycles in which text, besides being read and reread, must be typed and retyped, whereas text stored in the computer is never wholly retyped but only updated. As a result, the number of typographical errors is reduced and less proofreading is required. The most obvious advantage offered by on-line text-editing, therefore, is the significant reduction in the time required to produce a final document.

Most of the existing computer-assisted text-editing systems still use typewriter terminals--such as the ASR33 and the IBM 2741--as editing terminals, principally because of their low cost. Typewriter terminals are well suited for program editing, initial input of files, and minor editing of files. For tasks such as the editing of manuals, proposals, and other documents, in which many updates are necessary and revision time is at a premium, typewriter terminals begin to lose their usefulness because of their low output rate--about 15 characters/second. Moreover, the method of specifying edit operations, that is, the edit function to be done and the identification of the portion of the text to which it is to be applied, is also time-consuming.

When a CRT display, such as the IBM 2250 or the IBM 2260, is used as an editing terminal, a number of advantages ensue. One advantage is that the text is displayed at electronic speed, thus allowing many lines to be displayed at each stage of the editing process. Another advantage in using the display unit is that the identification of the text in question can be made easily either directly, by using the light pen, or indirectly, by using a "cursor". A third advantage of using a display unit for editing purposes is that edit functions can be selected by merely pressing the appropriately labelled "function keys", thus eliminating the necessity of typing command words. Also, when using a display terminal, the user would see a number of lines at a time

and, therefore, could think out and implement his changes simultaneously.

One of the best examples of a CRT-based text-editor is the Hypertext Editing System (Carmody et al, 1969), developed at Brown University for an IBM 2250. It allows flexible input and on-line editing. A light pen and a set of "function keys" under program control are used to indicate to the system the nature of the editing to be performed. In addition, many options are available so that text may be formatted both for on-line display and printouts.

Although the existing on-line editing systems provide many advantages over the hard-copy method, they are also relatively expensive due to the fact that each edit function is performed by the CPU. This disadvantage can be overcome by using terminals with local computing power, so that a section of the file can be transmitted to a terminal, be locally edited, and then be transmitted back to the main frame.

The present text-editing package at the University of Alberta uses, as editing terminals, the IBM 2741's and the IBM 2260's which are directly connected to the IBM 360/67 computer. Since the GRID has three core-memory banks and some computing power, it could provide local editing facilities. By using the IBM 2741's and the IBM 2260's for program editing, initial input, and minor editing of files, and the GRID for the editing of manuals, proposals, and other documents, the present on-line text-editing system could be made

less expensive. Where massive editing is involved, using a text-editing supervisor developed for GRID would reduce the number of interrupts to the 360 by an order of magnitude.

Use of the graphics CRT would have one further advantage, in that scientific documents almost invariably contain graphs or diagrams, as well as alphanumeric text. One can visualise a comprehensive document-editing system making use of two GRID supervisors, one for editing text and the other for diagrams. In this Chapter, however, we restrict ourselves to the text-editing supervisor.

## 5.3 Structure of the Text-Editing Supervisor

The text-editing supervisor (hereafter called "the text editor") is designed to allow the console user to input text and edit structured files through manual input devices. The duties of the display computer would be:

(1) to accept text input from the alphanumeric keyboard;

(2) to maintain and display a temporary text buffer;

(3) to perform editing functions;

(4) to display files transmitted from the main computer to the GRID terminal; and,

(5) to transmit messages to the main computer.

The 360 would be used for storing and transmitting display files. It would also be used for making changes in the stored files,

interpreting the messages from the GRID, and performing garbage collection required as a result of editing the text buffer.

The supervisor provides three modes of operation--the input mode, the edit mode, and the command mode--each of these modes designated by a specific function key. The function keys assigned for these purposes are enabled throughout, so that switching between modes is possible.

### 5.3.1 The Input Mode

Initial input of files would usually be done from punch cards, or through IBM 2741 teletypewriters. However, the input mode allows the user to append files of text at the display console through use of the alphanumeric keyboard. By pressing the key assigned as the "input key", the user has access to the temporary buffer in order to input textual information.

The textual information would be considered as a string of words, a word being a string of characters excluding blanks. This group of characters, or word, would be treated as a unit when the display file is organized or edited.

The temporary text buffer for the temporary file is to be long enough to hold about six pages of text, where a "page" is thirty lines on the display screen, or 2000 characters. When inputing text, each page, once completed, is saved in the display computer and

the user is automatically supplied with a blank page. This process continues until the buffer becomes full, at which time the text buffer is sent automatically to the 360 for addition to the "active file". However, the text buffer could be sent to the 360 at any time, at the discretion of the user, by pressing the BUFFSEND key.

The user would have a number of files of text. These files are stored in the auxiliary memory (disc) of the main computer. Files of text are stored as variable-length records; however, once the text is in the core, the lines are treated as fixed in length and padded to 86 characters with blanks. Each file has a name for identification purposes, giving the user access to previously stored files for reviewing or for re-editing.

The user's first responsibility when using the input mode is to activate a file through the "command mode" (to be discussed). In the case in which he wishes to edit or extend an existing file, the user activates that particular file by issuing a command and identifying the file by giving its name. However, if he wishes to create a new file, the user specifies in command parameters the name and the intended size of the file.

### 5.3.2 The Edit Mode

The edit mode is used for modifying existing files. As mentioned earlier, editing is done entirely in the display computer

except for a very few operations, such as the deletion of a series of pages of a file, which is done by the 360 computer.

Editing is done on a page-by-page basis. To edit a page of a particular file, the console user requests its transfer from the main computer to the display terminal. He does this in the command mode by specifying the name of the file and a string (enclosed in single quotes (" ' ") that is scanned for in the file as parameters. Commands to perform edit functions are executed as they are entered, so that the user is seldom forced to wait for system response.

Editing procedures could be classified under four headings--delete, insert, replace, and scan. A different function key is used to perform each editing operation.

To delete, the console user pressis the DELETE key, and uses the light pen to indicate the end points of the word, or words, to be deleted.

To replace, the console user presses the REPLACE key, types the replacement word(s) which appears in the edit area of the screen, and indicates with the light pen the end points of the word(s) to be replaced.

To insert, the console user presses the INSERT key and types the text to be inserted which appears in the edit area of the screen. Then he indicates with the light pen the word which the insertion is to follow.

To scan, the console user presses the SCAN key, and types a string

that is scanned for in the text buffer. If the string is found, the line which contains it becomes the current line.

### 5.3.3 Command Mode

The command mode is used for communication with the main computer. It allows the console user to construct messages for transmission to the 360. To do this, the console user, once in the command mode, constructs a message by typing the commands and their parameters. For example, to give a command "delete file YY", the user types

DELETE YY

The messages transmitted to the 360 include commands such as:

--activate file YY

--create a new file YY

--delete file YY

--print file on printer

--display next page of file

A FORTRAN program using the GRIDSUB (Jackson, 1971) interprets these commands. In case of an illegal command, an error response is displayed. The SEND key is used to send messages to the 360.

## 5.4 Features of the Display Unit

The screen of the GRID for text-editing purposes is divided into three areas--text, edit, and message--as shown in figure 5-1. The message area, the top two lines of the screen, is used for displaying messages to be sent to the 360, and also for display of

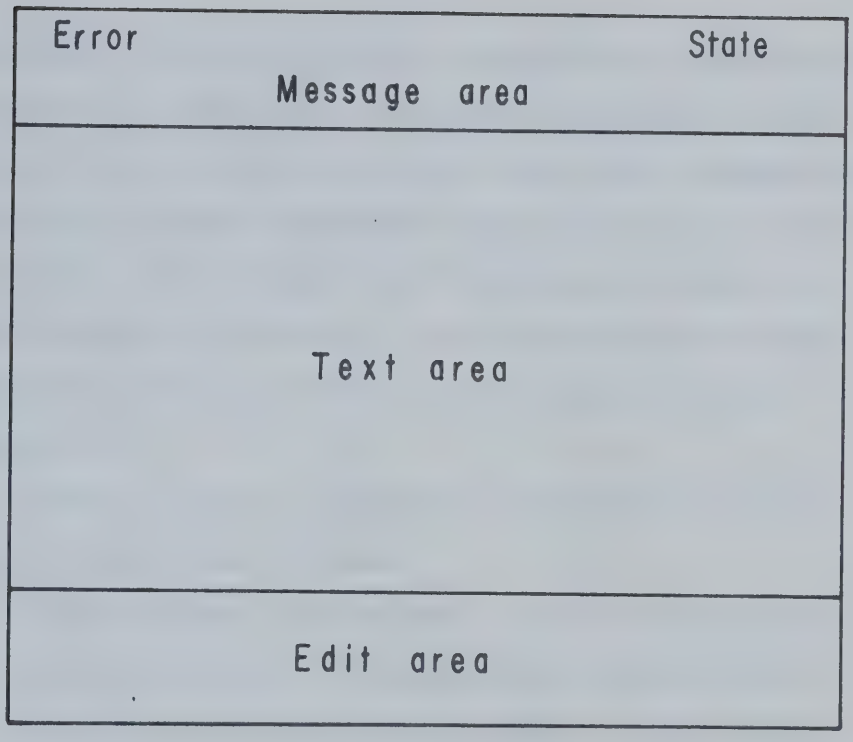



FIGURE 5-1

status and error messages. The edit area, the three lowest lines of the screen, is used for insertion and replacement purposes. The text area, the remaining portion of the screen, is used to display the text under construction, or a page of previously constructed text.

The user enters information on the text area by using the alphanumeric keyboard. As characters are input, they are displayed on the line in the text area where the "cursor" is

displayed. The cursor, which is a typing pointer, moves forward automatically as characters are entered, indicating the position that the next character would occupy on the line. The number of characters, including blanks, that could be entered into a line is limited. The "carriage return" key advances the cursor to the beginning of the next line. Typing errors are corrected by backspacing the cursor to the error and retyping the correct character(s).

Additional keys are designated on the keyboard to perform the following special functions:

CLEAR--This key, when depressed, wipes out the buffer and positions the cursor at the beginning of the first line.

LINE-UP--This key positions the cursor one line up from the current location at the beginning of the line. If the cursor is already at the top line the action is ignored.

BACKSPACE--When this key is depressed, the cursor backspaces one position, wiping out the last character entered. If the cursor is at the beginning of a line, use of the BACKSPACE key has no effect.

SPACE BAR--The SPACE BAR advances the cursor one position and enters a blank. If the cursor is at the end of a line, the SPACE BAR has no effect.

RETURN--The RETURN key advances the cursor to the beginning of the next blank line. If the cursor is already at the last available line, the key has no effect.

LINE CLEAR--The LINE CLEAR key clears the line containing the cursor and repositions the cursor at the beginning of the same line for reentry of text. If the cursor is positioned at the beginning of a blank line, the LINE CLEAR has no effect.

LINE SKIP--The LINE SKIP key causes the next line to be skipped and advances the cursor to the beginning of the succeeding line. If the cursor is already at the last line or the one before that, the LINE SKIP key has no effect.

### 5.4.1 Positioning the Cursor

If the cursor is not being displayed at the desired position, the user could position it with either the alphanumeric keyboard or the light pen. To do positioning with the keyboard, the user depresses the LINE-UP or the carriage return key to move the cursor up or down from its present position to the beginning of the desired line. To do positioning with the light pen, the user activates the light pen on any portion of a displayed text in order to move the cursor to the entry area associated with that request. Once the cursor is displayed at the desired entry area the user could position it more accurately by using the SPACE BAR and the BACKSPACE key of the alphanumeric keyboard.

## 5.5 Economic Justification for GRID Text-Editing

In this section, we describe a research procedure, and its results, which was done in order to provide a reasonably accurate estimate of dollars per hour of CPU cost that could be saved by using the GRID for text-editing.

The first step of this research procedure involved performing several hours of editing activity on an IBM 2741 terminal, and analyzing the edit functions performed on several old drafts of manuscripts developed by other users. From this, the frequency of occurrence of the most commonly used edit functions was estimated. Table 5-1 shows the frequency of use, in percentage ratio, of these edit functions.

| COMMAND | FREQUENCY IN PERCENTAGE |
|---|---|
| CHANGE | 34 |
| PRINT | 20 |
| INSERT | 15 |
| DELETE | 11 |
| SCAN | 10 |
| REPLACE | 8 |

Table 5-1

The second step of this research procedure was to determine the average CPU time required to perform each of the commonly used edit

functions. This was done by creating a file of 1500 lines of text, and performing each edit function ten times on different portions of the file. For example, the edit function, CHANGE, was performed at ten different places on the file by loading the "file editor" (*EDIT), making the change, and stopping the procedure. In each case, the time taken was recorded. From these recorded times, the average time of loading the file editor was subtracted. The average CPU time required to make a change was then calculated by averaging the resulting times. The same steps were carried out for each of the edit functions. Table 5-2 shows the average CPU times required to perform the different edit functions.

| COMMAND | AVERAGE CPU TIME (in seconds) |
|---|---|
| CHANGE | 0.086 |
| INSERT | 0.091 |
| SCAN | 1.122 |
| DELETE | 0.062 |
| REPLACE | 0.077 |
| PRINT(ONE LINE) | 0.034 |

Table 5-2

On the basis of the information given in Tables 5-1 and 5-2, for one hour of CPU time the average number of occurrences of the

GRID TEXT-EDITING SAVINGS

| (1) COMMAND | (2) AVERAGE NUMBER OF OCCURRENCE | (3) AVERAGE CPU TIME PER OCCURRENCE (in seconds) | (4) CPU TIME SAVING PER OCCURRENCE (in seconds) | (5) TOTAL CPU TIME SAVING (in seconds) | (6) CPU DOLLAR SAVING | (7) ACTUAL SAVING (in dollars) |
|---|---|---|---|---|---|---|
| CHANGE | 7344 | 0.086 | 0.086-0.034 = 0.052 | 0.052 x 7344 = 381.89 | 31.82 | 20.74 |
| PRINT | 4320 | 0.034 | 0.304-0.034 = 0.0 | 0.034 x 4320 = 146.88 | 12.24 | 1.73 |
| INSERT | 3240 | 0.091 | 0.091-0.034 = 0.057 | 0.057 x 3240 = 184.68 | 15.39 | 6.39 |
| SCAN | 2160 | 1.122 | 1.122-0.034 = 1.088 | 1.088 x 2160 = 2350.09 | 195.84 | 109.94 |
| DELETE | 2376 | 0.062 | 0.062-0.034 = 0.028 | 0.028 x 2376 = 66.53 | 5.54 | -3.06 |
| REPLACE | 1728 | 0.077 | 0.077-0.034 = 0.043 | 0.043 x 1728 = 74.30 | 6.19 | 0.53 |
| | | | | TOTAL: | 267.02 | 136.37 |

Table 5-3

various edit functions was calculated. Column 2 of Table 5-3 shows these figures. Column 3 shows the average CPU time required by each edit function per occurrence as already determined. Column 4 gives the CPU time saving per occurrence achieved by obviating the printing of responses, obtained by subtracting the average time taken for printing one line from the average CPU time required by each occurrence. Column 5 gives the total CPU time saving for each edit function per CPU hour. This was calculated by multiplying the average number of accurrences of edit functions with the CPU time saving per occurrence. Column 6 gives the CPU dollars saving for each edit function, calculated on the basis that one hour of CPU time costs three hundred dollars.

The amounts given in Column 6 would be the actual savings if the files were edited and held independently by the GRID. However, since the GRID has a small amount of core memory, only a portion of a file (about 80 lines) can be held and edited in it at a time.

Editing of files larger than eighty lines would involve a number of transmissions from the 360 to the GRID, depending on the size of the file and manner in which the file is being edited. For example, let us suppose that a user makes a change on a line in the portion of the file which is already in the GRID, then inserts a line at a position one hundred fifty lines ahead, and finally inserts a line next to the line where the original change was made. This whole

procedure would require two transmissions--the first transmission to get the portion of the file which includes the line one hundred fifty lines ahead, and the second to get back the portion of the file where the original change was made. However, if the user makes the change and the insertion to the next line first, and then inserts a line one hundred fifty lines down, only one transmission would be required in order to get the portion one hundred fifty lines down. Each transmission of a portion of a file from the 360 to the GRID would involve some CPU time. The figures given in Column 6 do not make any allowance for the cost of transmission and, therefore, overstate savings. To correct this, the old drafts were analyzed to determine how often each of the more commonly used edit functions was performed within a portion of a file that could be held by the GRID. In this respect, the initial drafts were compared to the drafts which led towards the final manuscripts.

It was found that, with regard to the edit function CHANGE, 90% of the times it occurred in the initial drafts was within a range that could be held by the GRID. It was also found that towards the final drafts only 75% of the times that a CHANGE occurred were within the range of the GRID. Similar figures were obtained for each of the edit functions. Column 2 of Table 5-4 shows the percentage of occurrence of each of these functions in the initial drafts.

Column 3 shows the percentages of occurrence toward the final drafts. The averages of these percentages are given in Column 4 of Table 5-4.

AVERAGE OCCURRENCE OF EDIT FUNCTIONS FREE OF TRANSMISSION

| (1) COMMAND | (2) INITIAL DRAFT | (3) TOWARDS FINAL | (4) AVERAGE |
|---|---|---|---|
| CHANGE | 90% | 75% | 83% |
| PRINT | 78% | 65% | 72% |
| INSERT | 75% | 60% | 68% |
| SCAN | 68% | 52% | 60% |
| DELETE | 80% | 35% | 58% |
| REPLACE | 70% | 55% | 62% |

Table 5-4

These averages represent the average percentage of occurrence of each edit function that would not involve any transmission. They, therefore, indicate those proportions of the values given in Column 6 of Table 5-3 that would be saved if the GRID was used for text-editing, apart from transmission costs.

A transmission of eighty lines of text to the GRID and back to the 360 takes 0.105 CPU seconds. This was determined by transmitting a text buffer of eighty lines to the GRID and back to the

360, and recording the time by using a system subroutine. The transmission time and cost associated with each edit function for one hour of CPU time are given in Table 5-5.

TRANSMISSION COST ASSOCIATED WITH EDIT FUNCTIONS

| (1) COMMAND | (2) TIME (in seconds) | (3) COST (in dollars) |
|---|---|---|
| CHANGE | 133 | 11.08 |
| PRINT | 126 | 10.50 |
| INSERT | 108 | 9.00 |
| SCAN | 90 | 7.56 |
| DELETE | 103 | 8.60 |
| REPLACE | 68 | 5.66 |

Table 5-5

These were calculated by determining from the values given in Column 4 of Table 5-4 the average percentage of occurrence of each edit function that would involve transmission. Then, using this percentage, the number of occurrences involved in a transmission and the number of transmissions per edit function, the total CPU time and cost involved in transmission associated with each edit function were estimated, and are shown in Columns 2 and3 of Table 5-5.

These costs were subtracted from the CPU dollar savings given in Column 6 of Table 5-3 to get the actual savings. These actual savings are given in Column 7 of Table 5-3. While allowance for transmission has reduced substantially the indicated savings, the amount of actual saving is still significant in terms of CPU dollar costs.

It would be desirable now to estimate the cost involved in using the GRID for one hour. The cost of equipment is 90,000 dollars, and the cost of maintenance per month is 450 dollars. The life expectancy of the GRID unit is at least five years. Therefore, the cost of equipment per month would be the unit cost divided by the number of months in its life expectancy plus the monthly maintenance cost. This comes to 1,950 dollars. Assuming that the GRID is used an average of eight hours per day, twenty-six days per month, the cost of the GRID per hour would be approximately 10 dollars.

To establish estimates of terminal costs requires an estimate of terminal operation, as distinct from the CPU hour used above. Further, since the GRID terminal would be notably faster than the 2741 terminal, we require separate estimates for these two alternatives.

Observation of actual editing sessions indicates CPU time per hour on the 2741 as being about one and a half minutes; equivalently each CPU hour of editing would require 40 hours on the 2741. With

a connect charge of $1.50 per hour this would cost $60.00.

Experience indicates that usage of the GRID would reduce terminal time by about 60%. On the above figures this would reduce terminal time to 16 hours, which would cost $160.00. Thus the net savings per CPU hour would be:

| | |
|---|---|
| Savings on CPU costs | $136 |
| Plus savings on 2741 costs | $ 60 |
| | $196 |
| Less cost of GRID | $160 |
| Net saving | $ 36 |

The resulting saving is very modest in the overall context, and, because of the various estimates on which it is built, subject to a considerable tolerance. However, in addition to any such direct savings there is the considerable saving of use time--a matter of 60%, or 24 hours out of 40 hours in this case. The value of this depends on the user. For a secretary this would yield $60.00, for a professional person considerably more.

## 5.6 Implementation: Ideas and Suggestions

From the point of view of implementation, a text-editing supervisor would be similar in several ways to a drawing supervisor; and, like the drawing supervisor, could be obtained by modifying and extending the skeletal general-purpose supervisor.

The functions provided in the general-purpose supervisor are primarily for the assembly of the operator's actions into a

message for transmission to the 360. It would, therefore, be desirable to include all these functions in the text-editing supervisor. Since they employ most of the function keys under "status 0", it would be desirable to assign the function keys to another status for text-editing purposes.

The edit functions could, therefore, be implemented by using each particular function key under the assigned status to perform a specific edit function. For example, function key 1 under the assigned status, "status 1", could be used to perform the edit function "delete". Corresponding interrupt routines would have to be added to the interrupt tables. It might be wise to begin the implementation of the edit functions by allowing a single edit function at a time and checking out the procedures.

## 5.7 Desirable Extensions

A feature that would be worth considering as an extension to the text-editing supervisor would be to allow the user to include "formatting codes" in the text. These would determine such things as margins, headings, paragraphs, indents, etc., to be interpreted at the time of printing the text on the printer. A further extension could allow for type-face variety, if a suitable microfilm or other printer were available.

# CHAPTER VI

# LANGUAGE PROCESSOR FOR CONSOLE COMMAND LANGUAGES

## 6.1 Introduction

During communication at the graphic console, the display operator constructs a message by performing a series of actions, such as picking displayed items with the light pen, pressing function keys, typing strings of alphanumeric characters, and drawing vectors between points on the screen. Each action involves an interrupt to the display terminal. Since no action is required by the 360 until the message is complete, the complete message could be checked for syntax specification and assembled by the display processor before interrupting the main processor. The advantage is that the number of interrupts to the main processor is minimized, thereby improving the response time and providing for more economical use of the 360.

The primary concern in this chapter is to define the general form of a console command language, and to describe a corresponding language processor which allows for the checking of inputs so that only syntactically correct messages are sent to the 360.

## 6.2 Console Command Language

Joyce and Cianciolo (1967) have said that: "A console command language is considered to be one in which communication at the CRT is carried out by continuous interaction with the pictorial repre-

sentations of a problem". This language consists of a series of console commands, such as "draw with light pen", "push buttons", "make light pen references to the objects on the screen", and "type numbers and names". Since this language is most natural to the user, it must be oriented to a specific application. Thus many console command languages may exist, each for a particular application. Examples of these user-oriented language systems are MIT SKETCHPAD (Sutherland, 1963), SKETCHPAD III (Johnson, 1963), and SDC GPDS (Vorhaus, 1966). However, these console command languages have common requirements, the most important of which are facilities for handling attentions from console input devices, assembling console commands into a message, and analyzing the assembled message for the construction and modification of the display file.

### 6.2.1 Formal Definition

A formal syntactic definition of the console command language as given by Morrison (Morrison, 1967), using Backus Normal Form (Naur et al, 1960), is outlined below.

<GRAPHIC PROGRAM> ::= <GRAPHIC STATEMENT> | <GRAPHIC PROGRAM> <GRAPHIC STATEMENT>

<GRAPHIC STATEMENT> ::= <TERMINAL SYMBOL> <TRAILER> | <GRAPHIC STATEMENT> <TERMINAL SYMBOL> <TRAILER>

<TRAILER> ::= <PART> | <TRAILER> <PART> | <NULL>

The elements of the terminal vocabulary (denoted by <TERMINAL SYMBOL>) must be defined for a specific application. The portion of the non-terminal vocabulary that is left undefined, <PART>, depends for definition upon the application. It consists of such things as <LIGHT PEN COORDINATE PAIRS> or <DISPLAY ITEM REFERENCE LISTS>.

Morrison defines the console command language in very general terms. Therefore, an extra notation is required in the definition to fit the peculiarities inherent in the nature of the language.

### 6.2.2 Notations

The input of the command language can come in a variety of ways, such as pen picks, pressing of function keys, input of text through alphanumeric keys, and indicating positions on the screen with the light pen. Therefore, the input source should be distinguished for each terminal and non-terminal symbol. The following notation is added to the metalanguage used in the definition for this purpose.

↑ - an arrow directed upwards is used to indicate that components must be picked with the light pen.

↓ - an arrow directed downwards is used to designate that items must be input by pressing function keys.

→ - a horizontal arrow directed towards the right is used to show that the required items must be given by the alphanumeric keys.

These symbols precede the terminal or non-terminal symbols to which they apply. Table 6-8 contains the syntax specifications of the console command language of the Campus Planning System (Deecker, 1970) using the BNF extended to include this notation.

### 6.2.3 Elements of a Console Command Language

The following are the basic elements of a console command language:

- a message to the 360 contains a series of statements or a single statement.
- each statement begins with a command word.
- the command word is either picked by the light pen or given by using a combination of a function key and a status key.
- if a message has more than one statement, each statement is ended by an end-of-statement character, except for the last statement of the message which is ended by the depression of the SEND key.

## 6.3 Language Processor

The language processor is designed to deal with the language input from the graphic console. It is concerned with the syntax of the language, but not with the meaning (semantics). Since the processor would be used for more than one application, it must be independent of any console command language. Therefore, a syntax-

directed recognizer/analyzer (Cheathman and Sattley, 1967) is used.

The analyzer interacts with the user on an element-by-element basis in order to restrict the user's choice of input symbols to those which are syntactically valid. Figure 6-1 shows how the analyzer and its associated tables fit in with the rest of the system. These tables are generated by a syntax pre-processor (to be discussed).

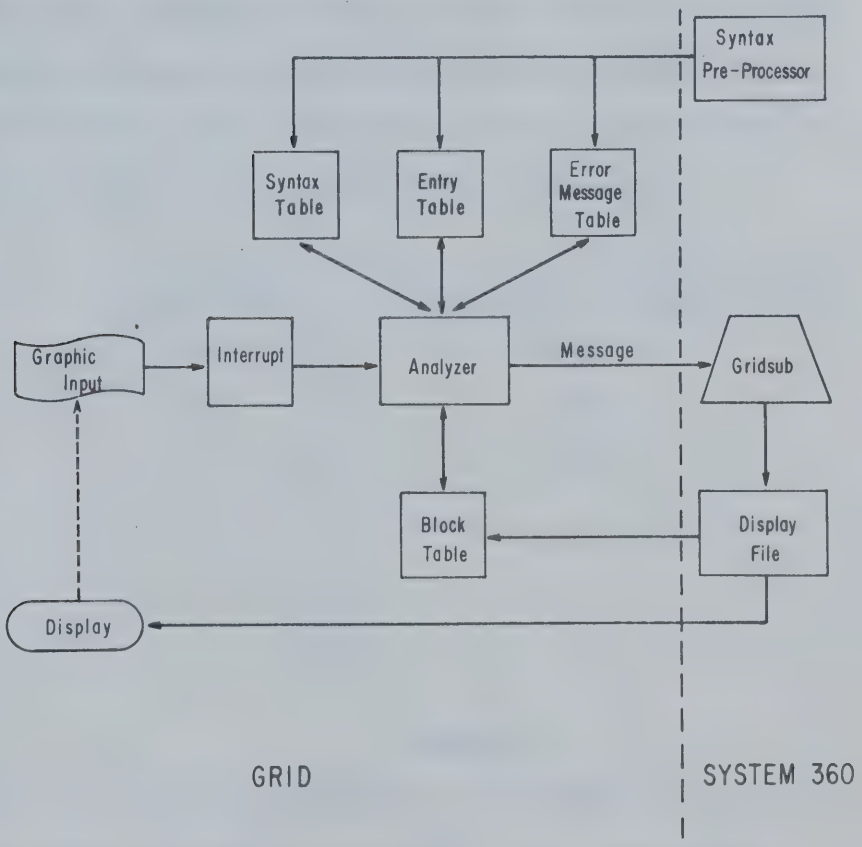



FIGURE 6-1

## 6.4 Analyzer Tables

The tables used by the analyzer are the "block table", the "entry table", the "syntax table", and the "error message table".

### 6.4.1 Block Table

The Block Table contains the block numbers and the starting buffer addresses of all the blocks (Jackson, 1971) present in the display file. Whenever an item is picked by the light pen, the analyzer searches the Block Table in order to determine which block has been picked. The contents and the logical structure of the Block Table are illustrated below by Table 6-1.

| BLOCK NUMBER | BUFFER ADDRESS OF START OF BLOCK |
|---|---|
| 100 | 2000 |
| 50 | 2156 |
| 71 | 2277 |
| . | . |
| . | . |
| . | . |

TABLE 6-1

Each time a message is transmitted to the main processor, the Block Table is transmitted along with it to be kept updated. The updating of the Block Table is done by the GRIDSUB.

6.4.2 Entry Table

The Entry Table is a list of addresses of command words defined in the Syntax Table. The Entry Table helps the analyzer to determine whether or not a statement begins with a command word; and, once this has been established, it also helps the analyzer to determine where it should begin to check the syntax specification of the remaining components in the Syntax Table. The contents and the logical structure of the Entry Table, using the Campus Planning System as an example, are shown in Table 6-2.

| | ADDRESSES |
|---|---|
| (COPY) | 401 |
| (ERASE) | 405 |
| (DELETE) | 407 |
| (JOIN) | 411 |
| (DRAW) | 414 |
| . | . |
| . | . |
| . | . |

TABLE 6-2

6.4.3 Syntax Table

The Syntax Table is the analyzer's key driving table. It holds the syntax specifications of a command language for a particular application. The Syntax Table allows the analyzer to do the syntax checking of a statement on an element-by-element basis. The contents and logical structure of the Syntax Table are illustrated below by Table 6-3.

| ADDRESS OF START OF ERROR MESSAGE DISPLAY CODE | TYPE | FKYVALUE | STVALUE |
|---|---|---|---|
| | BLOCKMIN | BLOCKMAX | IDRANGE |
| | | MINANCHAR | MAXANCHAR |

TABLE 6-3

The terms used in Table 6-3 are defined below:

TYPE- the number 512 indicating function key input, or the number 513 indicating alphanumeric key input

BLOCKMIN- the lowest block number of a block range used in the case of a light pen input

BLOCKMAX- the highest block number of the block range used in the case of a light pen input

FKYVALUE- a number from 0-to 9 indicating the function key value

MINANCHAR- a number indicating the minimum number of characters required in the case of alphanumeric input

MAXANCHAR- a number indicating the maximum number of characters allowed in the case of alphanumeric input

IDRANGE- two numbers, each ranging from 0 to 63--the first indicating the lower limit and the second indicating the upper limit--of the ID's assigned to the previously given block

STVALUE- a number, ranging from 0 to 63, indicating the status value

The end of the definition of each statement in the Syntax Table is indicated with a "-1". Whenever the analyzer encounters a "-1" in the Syntax Table, it expects the "end-of-statement" key or the SEND key to be the next input. If neither of these is the next input, the analyzer displays an error message.

### 6.4.4 Error Message Table

The Error Message Table consists of a list of error messages. The error messages are intended to inform the console user of the type of syntax error he had made during the construction of a statement. Some of the more commonly required messages remain permanently in the Error Message Table, others are composed by the applications programmer for his particular application. Whenever a syntactical error is made, the analyzer, using the starting address of the assigned error message given in the Syntax Table, locates and

displays the error message on the screen. This may be done simply by adding the error message to the display file as shown in figure 6-2.

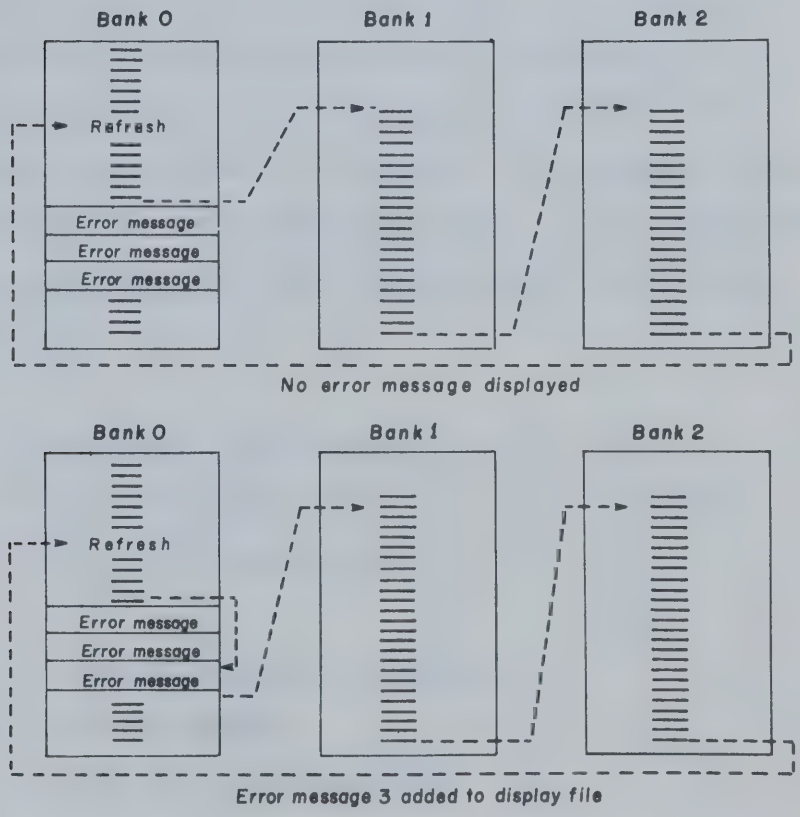



FIGURE 6-2

## 6.5 Syntax Pre-Processor

The syntax pre-processor is used to generate the contents of the Entry Table, the Syntax Table, and the Error Message Table. statements are input in a series of terminal symbols to the pre-processor. The first element of each statement is used to generate the contents of the Entry Table. The remaining elements are used to generate the contents of the Syntax Table.

The functioning of the pre-processor can probably best be explained through the use of an example. As our example, let us consider the first three statements of Table 6-8. The actual input to the pre-processor for these three statements is shown below in Table 6-4.

```
↑(1,2,2,22) ↑(2,40,60,0) #
↑(1,2,2,33) ↑(3,2,2,45) #        } Part A
↑(1,2,2,66) ↑(4,513,1,8) #

1- START STATEMENT WITH A COMMAND
2- POINT TO MAPNAME
3- POINT TO AVAILABLE/ON-SCREEN   } Part B
4- TYPE MAPNAME
# denotes the end of a statement
```

TABLE 6-4

Each component of the statements given in Part A has been defined according to the format given in Table 6-3, with one exception. This being that the "address of start of error message display code" has, in each case, been replaced by an error message number.

After receiving the input, the pre-processor first stores the error messages (Part B) in the Error Message Table in a sequential form. At the same time, it saves the starting address of each message for later storing in the Syntax Table.

The contents of the Entry Table, the Syntax Table, and the Error Message Table, as they would be generated by the pre-processor for this example, are shown in Table 6-5, Table 6-6, and Table 6-7, respectively.

| LOCATION | CONTENTS | ITEM |
|---|---|---|
| 200 | 600 | DISPLAY |
| 201 | 611 | LIST-MAPS |
| 202 | 622 | CREATE |

TABLE 6-5

| ITEM | LOCATION | CONTENTS | MEANING |
|---|---|---|---|
| DISPLAY | 600 | 1300 | ERROR MESSAGE ADDRESS |
| | 601 | 2 | BLOCKMIN |
| | 602 | 2 | BLOCKMAX |
| | 603 | 22 | IDRANGE |
| MAPNAME | 604 | 1325 | ERROR MESSAGE ADDRESS |
| | 605 | 40 | BLOCKMIN |
| | 606 | 60 | BLOCKMAX |
| | 607 | 00 | IDRANGE |
| | 610 | -1 | END OF STATEMENT |
| LIST-MAPS | 611 | 1300 | ERROR MESSAGE ADDRESS |
| | 612 | 2 | BLOCKMIN |
| | 613 | 2 | BLOCKMAX |
| | 614 | 33 | IDRANGE |
| AVAILABLE/ | 615 | 1336 | ERROR MESSAGE ADDRESS |
| ON-SCREEN | 616 | 2 | BLOCKMIN |
| | 617 | 2 | BLOCKMAX |
| | 620 | 45 | IDRANGE |
| | 621 | -1 | END OF STATEMENT |
| CREATE | 622 | 1300 | ERROR MESSAGE ADDRESS |
| | 623 | 2 | BLOCKMIN |
| | 624 | 2 | BLOCKMAX |
| | 625 | 66 | IDRANGE |
| MAPNAME 1 | 626 | 1355 | ERROR MESSAGE ADDRESS |
| | 627 | 513 | TYPE |
| | 630 | 1 | MINANCHAR |
| | 631 | 8 | MAXANCHAR |
| | 632 | -1 | END OF STATEMENT |

TABLE 6-6

| LOCATION | MESSAGE |
|---|---|
| 1300 | PT2 |
| | X |
| | Y |
| | START |
| | STATEMENT |
| | WITH A |
| | COMMAND |
| | JDD |
| | AAAA |
| 1325 | PT2 |
| | X |
| | Y |
| | POINT TO |
| | MAPNAME |
| | JDD |
| | AAAA |
| 1336 | PT2 |
| | X |
| | Y |
| | POINT TO |
| | AVAILABLE |
| | OR |
| | ON-SCREEN |
| | JDD |
| | AAAA |
| 1355 | PT2 |
| | X |
| | Y |
| | TYPE |
| | MAPNAME |
| | JDD |
| | AAAA |

TABLE 6-7

## 6.6 Operation of the Analyzer

The analyzer's operation is described as follows. Tracing the flow of input, the source item enters the system from a terminal and is routed to the analyzer. The analyzer compares it with the item(s) placed in the Syntax Table and, as a result of the comparison, the analyzer takes one of the following two actions:

(1) accepts the incoming input with a visible response if it conforms to the syntax specifications set out in the Syntax Table; or,

(2) rejects the incoming input with an error message if it does not conform to the syntax specifications set out in the Syntax Table.

Each accepted input is stacked in the message area until the message buffer is full or the SEND key is pressed. If the message composed by the operator exceeds the storage space available, an error message, such as "message too long", is displayed on the screen. The operator then has to reset the buffer and compose another, shorter message.

The depression of the SEND key transmits the contents of the message area as a message to the 360. However, if the SEND key is pressed in the middle of a statement, this action is ignored and an error message displayed.

To show the operation of the analyzer and its adaptability to use with various systems, let us consider two examples.

Example 1

As a first example, let us consider the operation of the analyzer with the CALD System (Johnson, 1969). The CALD System displays on the screen the logic block types (AND, OR, and NOT gates), as well as a set of command words.

The designer, working with this system and using the light pen, can exercise his design function by pointing to the logic blocks, command words, and positions on the screen. Now, if he wants to display a logic gate at position (X,Y) on the screen, he points to the command word COPY, one of the logic blocks, and then the (X,Y) position at which he wants the logic block to appear. According to our language specification this series of actions is considered a statement, whose syntax is:

<COPY> ::= ↑COPY↑ <LOGIC BLOCK> ↓ <X,Y>

Suppose that all of the command words are assigned to block number 2, and defined by an identification number. Let us assume that, in this case, COPY is denoted by block number 2 and ID 4. Again, let us suppose that the logic blocks are assigned to blocks 20 through 22 inclusive, and that positions (X,Y) are indicated with the light pen once the function key 1 in status 0 is pressed. Figure 6-3 is a functional representation of the portions of the analyzer's tables which are set up by the pre-processor to check the syntax specifications of this statement.

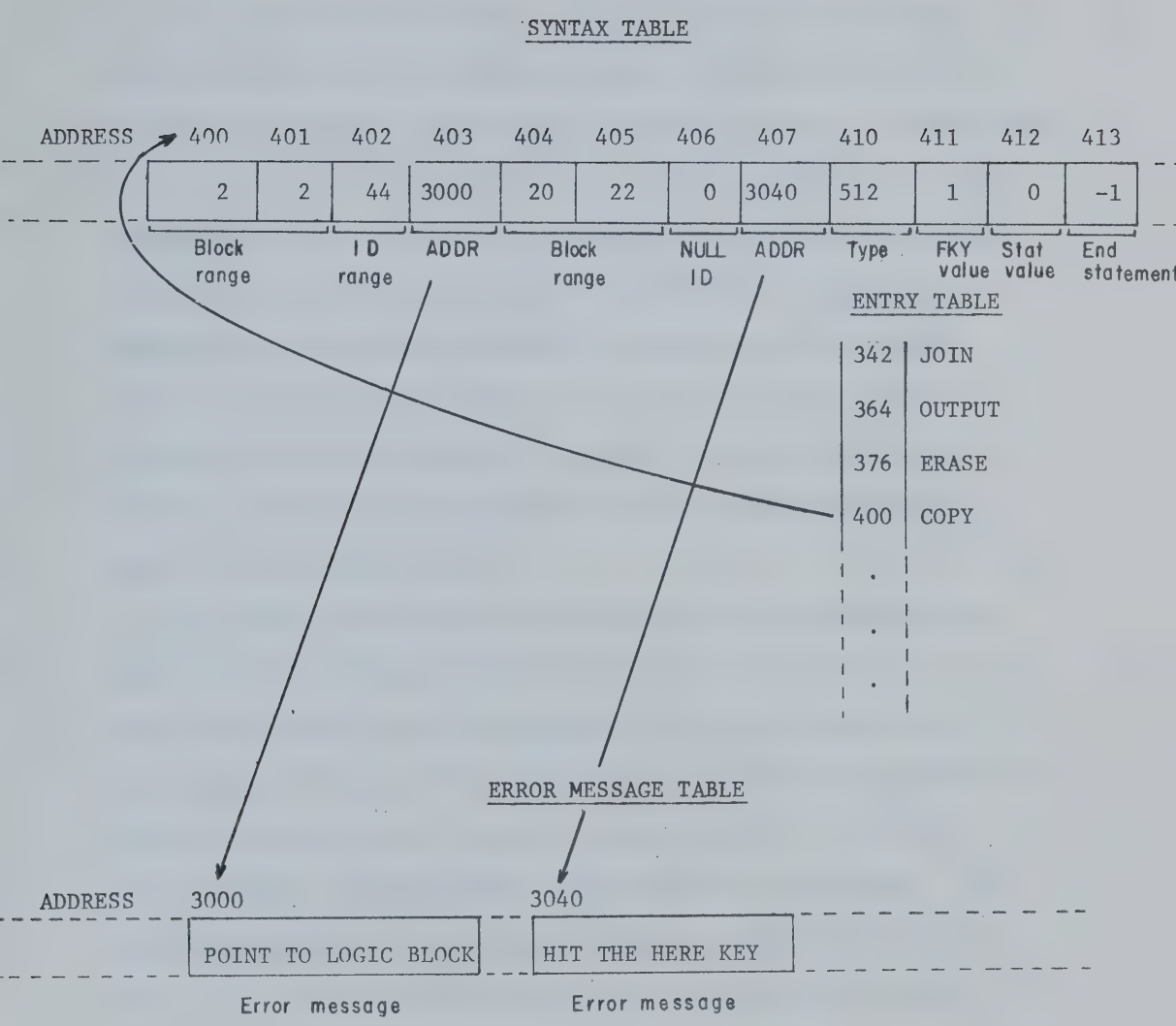



FIGURE 6-3

After communication has been established with the 360, the first action performed by the console user initiates the operation of the analyzer. In this case, the first action should be that of picking the command word COPY with the light pen. If this is done, the input

to the analyzer is block number 2 and ID 4. The analyzer, using entries given in the Entry Table, searches the Syntax Table until it finds the combination of block 2 and ID 4, or until it exhausts all the entries given in the Entry Table. In the case where the analyzer exhausts the list of entries, it assumes that the statement was not initiated with a command word; and, it, therefore, displays the appropriate error message. However, in this case, the analyzer finds the combination of block 2 and ID 4 in the Syntax Table at location 400. It then accepts this input, and allows the user to input the next part of the statement which it checks with the contents of the Syntax Table.

According to the syntax specifications set out in the Syntax Table, the only subsequent input that would be acceptable is that which is within the block range of 20 to 22. The only way this input can be received is if the console user picks a logic block type with the light pen. Any other action results in the error message "point to logic block" being displayed on the screen. Let us assume, however, that the console user does point to a logic block. The analyzer accepts this input; and allows the user to initiate further input.

The Syntax Table specifies that the next input should be given by the function key 1 in status 0. If the next input conforms to this specification, it is accepted by the analyzer. However, if the next input does not conform to this specification, it is rejected with an error message.

The next entry in the syntax is a "-1". This indicates the end of the statement. The analyzer expects the next input to be the end-of-statement character or the SEND key.

Example 2

As a second example, we can consider a command of the Campus Planning system (Deecker, 1970) which allows the planner to display a map, picked from a list of displayed map names, on the screen. To do this the user first points the light pen to the command word DISPLAY, then to the desired map name. In this system the command words are assigned to block number 2 and defined by an identification number. The command DISPLAY is defined by the block number 2 and identification number 2. The map names are assigned to block numbers ranging from 40 to 60. The syntax specifications for the Campus Planning System are given in Table 6-8. Here, this statement is encoded as follows:

<DISPLAY> ::= ↑DISPLAY↑ <MAPNAME>

A functional representation of the portions of the analyzer's tables which check the syntax specifications of this statement is shown in figure 6-4. The steps taken by the analyzer to perform the analysis of this statement are similar to those performed in the analysis of the statement in Example 1.

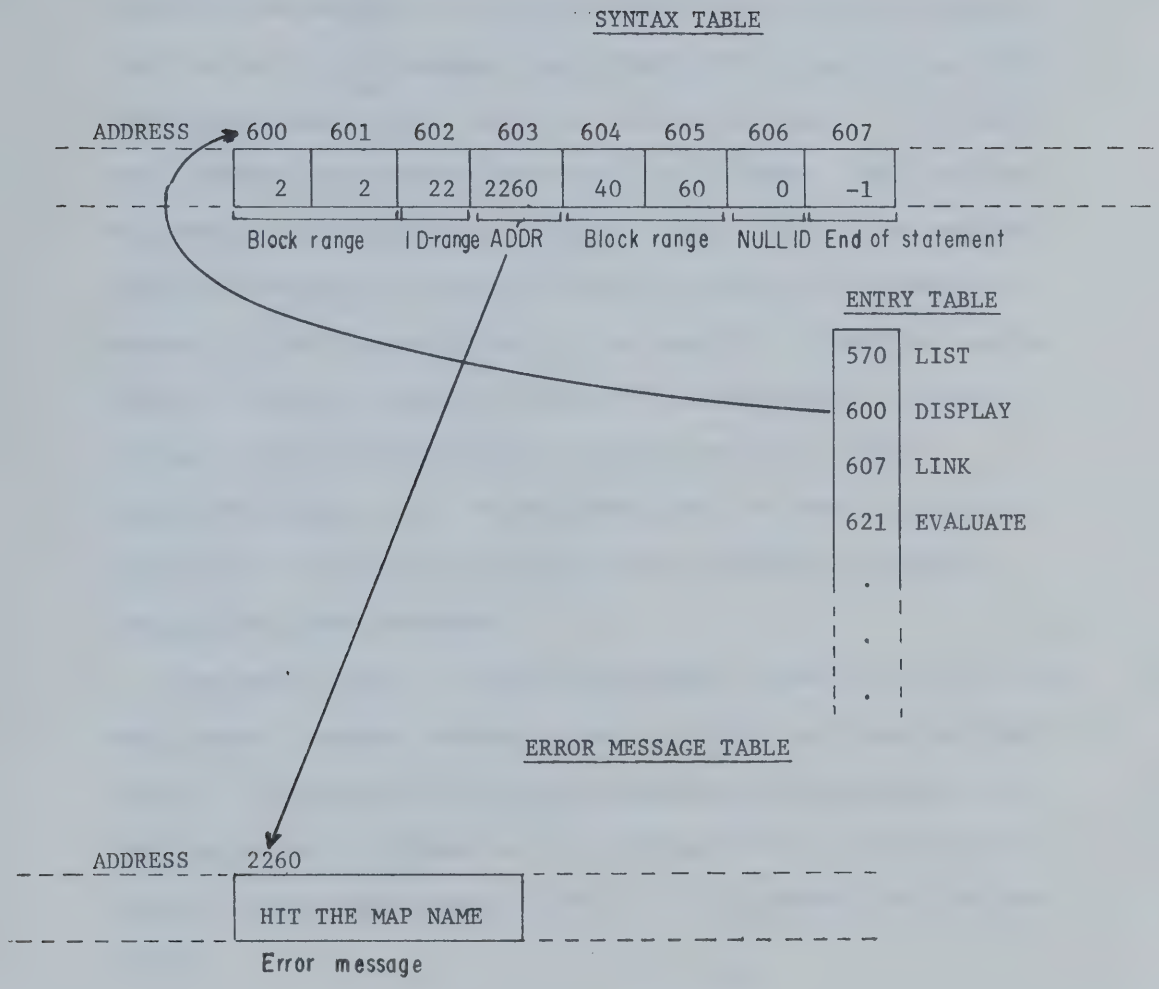



FIGURE 6-4

## 6.7 Conclusion

We have attempted to explain some of the principles of an imperative command processing system through the use of a syntax-directed analyzer. A very important feature in the design of this

system is the decision to perform maximum error-checking during the construction of the message, thus allowing only those messages which are syntactically correct to go to the main processor. The present most commonly used methods (Morrison, 1967) (Samuel, 1969) perform only minimal error-checking during the construction of the message, therefore leaving the burden of complete checking to the main processor. The evident advantage of the imperative command processing system is that it reduced the number of interruptions to the main processor, hence allowing more economical use of the 360. A second advantage is that the programmer does not have to code an analyzer for each application, but simply specifies the console language and error messages.

The major premise on which this system is based is that all console command language statements must fall into the class described earlier. If the user of the system accepts this restriction, his application can be implemented with minimal effort. He would, of course, compose the language by which he would communicate with the system.

<COMMAND> :: = <DISPLAY> | <LIST> | <DRAW> | <LINK> | <EVALUATE>
| <ERASE> | <SCALE> | <PLOT> | <OVERLAY> | <SHADE>
| <WINDOW> | <DELETE> | <CENTRE> | <RESTART> | <END>

<DISPLAY> :: = ↑DISPLAY ↑<MAPNAME>

<LIST> :: = ↑LIST-MAPS {↑AVAILABLE / ↑ON-SCREEN}

<CREATE> :: = ↑CREATE → <MAPNAME 1>

<LINK> :: = ↑CATENATE ↑<MAPNAME> ↑ <VECTOR>

<EVALUATE> :: = ↑EVALUATE ↑<MAPNAME>

<ERASE> :: = ↑ERASE ↑<MAPNAME>

<SCALE> :: = ↑SCALE {↑UP / ↑DOWN}

<PLOT> :: = ↑PLOT

<OVERLAY> :: = ↑OVERLAY ↑<MAPNAME>

<SHADE> :: = ↑SHADE ↑<MAPNAME>

<WINDOW> :: = ↑WINDOW →<INTEGER>

<CENTRE> :: = ↑CENTRE ↑<XY>

<DELETE> :: = ↑DELETE ↑<MAPNAME>

<RESTART> :: = ↑RESTART

<END> :: = ↑STOP

<MAPNAME> :: = <ABC> <ABC> <ABC> <ABC> <ABC> <ABC> <ABC> <ABC>

<ABC> :: = A |B |C..... |Z

<INTEGER> :: = 0 |1 |2 |..... |8

<VECTOR> :: = <VECTOR> ↓ <LINE> | ↓ <LINE>

<XY> :: = a point indicated with the light pen

<LINE> :: = that defined by a particular function key 2

<MAPNAME 1> :: = 8-character string on A/N keyboard

TABLE 6-8

# CHAPTER VII

# CONCLUSION

An often quoted sentence of Ronald L. Wigington (Wigington, 1967) states that "The combined man-machine system environment of an on-line graphical system is a trade-off between what the computer can do well and what the man can do better". The computer's sole justification for existence in this combined system environment is to extend the capabilities of man's intellect by presenting information to him in the form in which it can best be understood.

Computer graphics has been implemented within environments which involve a variety of hardware configurations. These hardware configurations have been discussed under two headings: "Directly-Coupled Displays"--in which the display unit is directly connected to a medium-to-large computer--and "Indirectly-Coupled Displays"--in which a medium-sized computer controls display regeneration and processes a large proportion of the operator's requests, but is coupled to a large computer which processes the major requests.

Although the indirectly-coupled systems exemplified by the IBM 1130-2250 have provided a better environment for interactive graphics than the old, directly-coupled systems, they do not provide the sort of environment within which interactive graphics pro-

grams can be most easily written and most economically operated.

From the outset of this project, a hardware configuration has been utilized in which the graphics computer plus CRT is considered to be a programmable terminal, operable under a large, time-sharing computer system. More specifically, the configuration used at the University of Alberta couples a Control Data 160A/GRID graphics terminal to an IBM 360 Model 67. The software support considered for this system has been discussed. It recommends a multisupervisor scheme for the display terminal. Under this scheme, a number of display supervisors could be designed, each providing the standard functional behavior required for a specific class of application. It, therefore, makes it easier for the applications programmer to use interactive graphics techniques by eliminating the necessity of having him program the display terminal for each application.

The functional behaviors thought to be most useful are discussed and designed. They include a drawing and editing supervisor, for which the structure and implementation along with light pen tracking methods have been described, a text-editing supervisor, and a language processor.

In this thesis the main consideration of the author has been to facilitate the applications programmer in the use of interactive graphics. The author believes this has been

achieved by eliminating the need for the user to program the display terminal as well as the large computer. Another primary objective has been to reduce the operating cost involved in on-line graphics, to the extent that it could be used as a standard tool in computing installations. Again the author feels that this objective has been met, for, by writing a supervisor tailored for a particular application, the display terminal can be used to perform actions for which the main computer would otherwise be required. As a further extension to this work, it would be desirable to implement a text-editing supervisor and a language processor. This could be done by modifying or extending the general-purpose supervisor.

REFERENCES

Appel A., Dankowski T.P. and Dougherty R.L. 1968. 'Aspects of Display Technology', IBM Systems Journal, vol. 7, No. 3.

Bourne S.R. 1971. 'A Design for Text Editor', Software--Practice and Experience, vol. 1, pp. 73-81.

Carmody, Steven, Gross W., Nelson T.H., Rice D., and Dam A.V. 1969. 'A hypertext Editing System for the /360', In Pertinent Concepts in Computer Graphics, M. Faiman and J. Nievergelt (Eds.) Univ. Illinois, Urbana Ill., pp. 291-330.

Davis M.R. and Ellis T.O. 1964. 'The RAND Tablet', A Man-Machine Graphical Communication Device AFIPS Conf. Proc. FJCC.

Deecker G.F. 1969. 'A Comparison of Methods for Digital Encoding of Arbitrary Line Figures', Computing Review, Univ. of Alberta, vol. 2.

Deutsch L.P. and Butler W.L. 1967. 'An Online Editor', Journal ACM, vol. 10, No. 12, pp. 793-799.

Elliott W.D., Warren A.P., and Dam A.V. 1971. 'Computer Assisted Tracing of Text Evolution', AFIPS Conf. Proc., FJCC 37.

Gray J.C. 1967. 'Compound Data Structures for Computer-Aided Design', Proc. of 22nd National Conference of the Association for Computing Machinery, pp. 355-365.

Hough M. 1968. 'Developing a Sense of Place for Community Colleges', Canadian University.

Huen W.J., Jacobsen F.B., May K.F., and Penny J.P. 1969. 'Computer Graphics for the Fortran Programmer', Ref. Manual, Univ. of Alberta, Edmonton.

Jackson W.C. 1971. 'Computer Graphics for the Application Programmer', Ref. Manual for the IBM 360/GRID Graphics Software System, Department of Computing Science, Univ. of Alberta.

Jacobsen F.B., May K.F., Huen W.H., and Penny J.P. 1970. 'Computer Graphics for the FORTRAN Programmer', Univ. of Alberta Computing Centre Publication.

Johnson B.V. 1969. 'Preliminaries to a Computer Aided Logic Design System', Computing Review, Univ. of Alberta, vol. 2.

Johnson T.E. 1963. 'SKETCHPAD III: A Computer Program for Drawing in Three Dimensions', AFIPS Conf. Proc., vol. 23, pp. 347-353.

Joyce J.D., and Cianciolo M.J. 1967. 'Reactive Displays: Improving Man-Machine Graphical Communication', AFIPS Conf. Proc., AJCC 31, pp. 713-721.

Morrison R.A. 1967. 'Graphics Language Translation with Language Independent Processor', AFIPS Conf. Proc., AJCC 31, pp. 723-732.

Samual I.N. 1969. 'Synthesis and Analysis: A Flexible Technique for Processing Command Language', IEEE Transactions on Computers, vol. c-18, No. 11.

Sulkers P., Smith H., and Zavitz, G. 1971. 'A Multibank Supervisor and I/O Software Package for Grid',Ref. Manual, Univ. of Alberta, Edmonton.

Sutherland I.E. 1963. 'SKETCHPAD: A Man-Machine Graphical Communication System', Lincoln Laboratory, M.I.T.

Vorhaus A.H. 1966. 'General Purpose Display System', Datamation, pp. 59-64.

Wigington R.L. 1967. 'Graphics and Speech Input and Output for Communication with Human', Computer Graphics, An Informatics Inc. Publication, Thompson Book Company, Academic Press, London.

## APPENDIX A

### Description of the DDYECT decoding routine

DDYECT (LINE, IX, IY, NUM, STX, STY, ZN)

### Parameters

LINE (No of lines): Mode INTEGER: Range $1 \leq$ LINE $\leq 20$

IX, IY (X,Y coordinates): Mode INTEGER vector; Dimension MAX

NUM (No. of points in each line): Mode INTEGER vector; Dimension MAX 1

STX, STY (starting points of each line): Mode INTEGER vector; Dimension MAX 1

ZN: N is a statement number

MAX: Mode INTEGER; Range $1 \leq$ MAX $\leq 2777$

MAX 1: Mode INTEGER; Range $1 \leq$ MAX 1 $\leq 20$